Woodstein: A Web Interface Agent for Debugging E-Commerce

by

Earl Joseph Wagner

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Masters of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

Author . . . .
Media Arts and Sciences
August 8, 2003

Certified by . . . .
Henry Lieberman
Research Scientist
Thesis Supervisor

Accepted by . . . .
Andrew B. Lippman
Chairman, Departmental Committee on Graduate Students

Woodstein: A Web Interface Agent for Debugging E-Commerce

by

Earl Joseph Wagner

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
on August 8, 2003, in partial fulfillment of the
requirements for the degree of
Masters of Science in Media Arts and Sciences

Abstract

Woodstein is a software agent that works with a user's web browser to explain and help diagnose problems in web processes, such as purchases. It enables the user to inspect data items in ordinary web pages, revealing the processes that created them. It provides an integrated view of the processes and data associated with a user's actions at a web site, and retrieves related information on the same web site, or even on different web sites. When the user inspects data that looks incorrect, Woodstein helps manage hypotheses about causally related data and processes that look incorrect and provides guidance in the process of elimination to isolate the unsuccessful process or wrong data.

Thesis Supervisor: Henry Lieberman
Title: Research Scientist

Thesis Committee

Thesis Advisor ....

Henry Lieberman

Research Scientist

Thesis Supervisor

Thesis Reader ....

Mitchel Resnick

Associate Professor

MIT Media Laboratory

Thesis Reader ....

Charles Rich

Distinguished Research Scientist

MERL Cambridge Research

Acknowledgments

I would like to thank Henry Lieberman for his patience and wise guidance as an advisor and, more importantly, as a mentor. I would also like to thank my thesis readers, Charles Rich and Mitch Resnick, as well as Marc Millier, Ben Grosof, Daniel Greenwood and Bill Mitchell for their advice, encouragement and enthusiasm.

Creating Woodstein has been a thrilling experience, and I'd like to thank those who I had the opportunity to work with. Chris Laux played an important role in shaping the interface over the course of many discussions, and his contributions to the implementation were significant. Last minute help from Vijay Shah and Hyunsuk Kim was greatly appreciated. Thanks also to my officemate Hugo Liu for never letting me get complacent.

My experience at the Media Lab was also greatly enriched by the students and faculty who cared.

Finally, I'd like to thank my family for giving me the support and encouragement I needed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Actions on the Web

Once upon a time, the web was primarily a medium for reading information, through browsing and searching. Things change. In addition to being an information source, the web is rapidly becoming a medium for performing the actions of everyday life that, in the past, used to require one's physical presence. Now, we routinely buy and sell things, arrange meetings, jobs and dates, even sign legal documents on the web. E-commerce itself depends upon being able to perform actions like these.

But the web's interface has remained stuck in the interface paradigms we inherited from information retrieval: following hyperlinks, scrolling, and using search engines. Web browsers are oriented toward sequential presentation of pages of information. They have, of course, been extended with the minimal tools required to perform actions, such as buttons, pop-up menus, text fields, and so on. But actions are more than just sequential pages of information, or even paths of hyperlinks.

Actions themselves are structured information. They are performed to accomplish goals. Goals have sub-goals. The data in pages is set while accomplishing goals. Today, none of that structure is explicitly represented in web browsers, but it is important for the user. Users need help answering questions about the operation of the actions they perform: "Where am I?", "What just happened?", "What do I need to do next?", "How did that data item get there?", "Is it correct?", "If something went wrong, how can I figure out where the problem is?", "How can I fix it?"

We are long past the point where these kinds of problems can be solved simply by "good web site design" or "user-centered design and testing methodology" or any kind of web standards. While these principles are indisputably important, and some web sites do provide some help for their own local actions, and standardization of interfaces undoubtedly would help, that's not the whole story. Increasingly, users are performing actions that involve multiple web sites, particularly in the domain of e-commerce. An on-line purchase may involve a merchant's web site, a payment service such as a credit card, and a shipper. A financial transaction often involves different institutions at each end. A trip reservation may involve airline, hotel and car rental web sites. The connections between these exist only with the user - no individual web site can relate them all. Even worse, a user be performing many simultaneous actions, and those actions may extend over long periods of time. It can be overwhelming just to keep track of multiple actions, and the importance of retrieving, compiling and managing all of this information only increases when something goes wrong.

1.1.1 Ongoing Actions

All of these difficulties consumers face in e-commerce are compounded when performing multiple, ongoing actions. It's enough of a challenge for consumers to remember and manage details related to one problem they're trying to resolve; it becomes even more frustrating to try to manage the details associated with multiple processes and the problems that arise.

As consumers are increasingly required to interact with vendors through the structured format of web pages, they would benefit from tools that can act as a guide in these interactions. By keeping track of the user's information and making it available at relevant times and locations, the inconvenience they face could be minimized

In some cases, an action may require many steps and be confusing. It can be tough for the user to see the big picture of the overall action, especially when it takes multiple days or is extended over multiple web sites. For example, new employees in an organization, or any employee using a new internal web site must learn how to perform associated actions such as setting up a corporate stock purchase plan, enrolling in training courses, and so on. Actions like these may require some steps to be taken initially, then, after some processing has been performed, some additional steps a few days later or on a different site. It would be helpful to be able to see a visualization of the current status of actions like these to see the steps that have been performed and the steps that still remain to be performed.

Increasingly, employees are interacting with different departments through a web-based front end to submit a request and monitor its status. When something goes wrong, the employee is left to call the number at the bottom of the web page and experiences the same frustration with internal support that consumers face with customer service.

1.2 Problems in E-Commerce

“Your call is important to us

“Please enter your 15-digit card number, your PIN...

“If you’re calling from a touch-tone phone, please press 6 for customer service now”

Electronic commerce is great - you log onto a web site, type in what you want to its search engine, fill out a simple form, give your credit card number, and the shipper brings you the stuff the next day. It’s simple, painless, efficient, and effective.

Except when it doesn’t work. People inevitably mistype numbers, misunderstand directions, or just click on the wrong buttons. Vendors lose orders, or confuse names, and computer systems crash. Then what?

More often than not, customers have to pick up the phone. And they often reach a phone tree where they’re forced to navigate through complex menus, a tedious process that might or might not give the right choice of problem to be solved. It might give another phone number to be dialed. It might ask for card numbers or transaction numbers that aren’t readily at hand, and have to be looked up offline. If someone in customer service is successfully reached, that person (often a low-paid worker in a high-pressure call center) may specify a tedious process to be performed. They may not be empowered to actually understand or fix the problem themselves. Customers find themselves bounced endlessly from one support person to another. All of us have had these kinds of experiences.

Customer service problems are incredibly frustrating. Not only do they cause frustration about the immediate transaction, they also poison the relationship between customers and vendors. Customers feel like they are being deflected, that they are not being listened to, and that vendors do not care about their time or about their relationships. This casts a pall over future interactions, in fact 80% of customers are less likely to buy online from a vendor again after a dissatisfying experience with its customer service[14].

Nobody expects technology to be perfect, and people are tolerant of mistakes and occasional failures, but what drives people crazy is when they get “stuck” - left with no recourse, no obvious way to systematically go from having a problem to understanding what is causing the problem and how to resolve it.

But it isn't all the fault of the vendors. The problem is difficult to fix simply though vendors spending more money on call centers, since human time, even if low paid, is expensive. Many businesses claim that, as soon as a customer picks up the phone to call about a problem, they have already lost money on that customer's relationship.

This problem is now becoming a major obstacle to further adoption of e-commerce itself. Many people who now choose brick-and-mortar shopping over e-commerce do so simply to "have a face to talk to" in case of problems. Again, the impact of the problem-resolution experience has a disproportionate effect on the trust relationship between a customer and vendor, and nowhere is the trust issue more important than in electronic commerce.

1.2.1 Confirmation Pages

The web browser, originally designed for viewing and authoring documents, shows its limitations most clearly with the confirmation page for an order. An online purchase is completed with the appearance of a confirmation page or receipt. Users are told to "print this for your records", though if they do then they lose the flexibility of having a digital copy. Even the more careful users who save a copy of the page, when this is possible [1], are left with no easy way to organize these receipts and have to create their own organizational systems.

1.3 Current Technologies for Problems in E-Commerce

A user goes onto the web to perform some goal, such as making a purchase. A purchase involves browsing and adding items to the shopping cart, then checking out, which involves providing shipping and payment information. Though the user thinks in terms of the overall goal, the browser and its associated technologies are only focused on the actual pages the user interacts with.

1.3.1 Tools for Managing User Data

At the very bottom level of a user's action are the details of a user's information, such as his address or credit card number. This is the focus of form-filling browsers and simplified

[1] often confirmation pages can't be reloaded, as saving a page requires in some browsers

authentication.

Form-Filling Browsers

Browsers provide some support for e-commerce by filling-in forms [13] [11]. This feature is helpful for users in managing multiple accounts with different user names and passwords. When the browser recognizes fields for addresses and credit-card information, it can also enter these automatically.

Simplified Authentication

There is some work within the computer software industry to standardize the processes of identification and authentication. Microsoft, with .NET Passport [12], and the Liberty Alliance[36], with their software aim to provide a single sign-on allowing users to enter personal information once and have it shared with the vendors they buy from automatically. Although this will eliminate many errors that currently result from customers supplying incorrect information, this still won't be effective for data that a user supplies for each interaction, such as purchase quantities.

Simplified Transactions

Some vendors, such as Amazon with their One-Click shopping, are working to simplify transactions. By minimizing the steps involved in making a purchase, techniques like One-Click prevent incorrect data from being entered. This can create problems if a user's data has changed, such as if the user's credit credit card has been updated, or the user has a new address.

Although a user's personal data are details that they should have help in managing, it's easy to see that these techniques work at a low level, and are quite fragile. For instance, one of these tools might keep track of a credit card, or an address, but what if the user uses multiple credit cards, or uses their addresses for work and home? It's clear that functionality is required for managing multiple roles, and as part of this, there is a need to be able to keep records of when data associated with these roles has been provided. This can be vital for diagnosing problems that appear later, or just for updating the different sites when a user's data changes, such as when the user moves.

1.3.2 Tools for Recording E-Commerce Transactions

Other solutions, such as Tower's WebCapture[22] focus on recording a user's interaction with a web site over the course of an e-commerce transaction. Although this is an important feature, it is only the beginning of the story. Understanding the cause of a problem may require understanding an entire process that spans multiple sites, or many pages on a single site. Often, transaction information appears on pages that weren't part of the original transaction, such as when the charge for a purchase appears on a credit-card transaction history page. Simply recording the pages the user interacted with isn't enough to assemble an integrated view of the transaction. For that, the system must be able to represent and reason about the transactions themselves. In order to allow the user to interact with high-level representations of transactions, it must assemble those representations itself.

1.4 Viewing E-Commerce Transaction Data at the Transaction Level

The user thinks of all of the web site interactions as part of a single transaction primarily, not separate interactions on disparate web sites. A system for diagnosing e-commerce problems should therefore allow the user to manage these transactions at the level of the transaction, not at the level of the individual sites involved. One possible solution exists in the idea of audit trails for data on the web. When a users sees some data in a page that looks incorrect, it would be helpful to be able to inspect the data and see an audit trail with the history of how it was created.

1.4.1 Audit Trails

Audit trails originated in business to record the accesses and changes of a record, and are also used it computer security to track intrusions into a system and changes that are made. In the domain of e-commerce, web sites don't typically offer audit trails for the objects they describe. For example, a shipper knows about the circumstances in which it received and also delivered a package, but that's only a slice of the history of the ordered items. A complete audit trail must provide an complete history of an object generated by integrating the histories described by multiple sites.

Audit trails are invaluable in business for understanding the history of some object or data item. Unfortunately, they are not currently available to users for the data they see in web pages.

1.5 Debugging E-Commerce

Our solution is to provide users with an E-Commerce Debugging Tool that enable users to diagnose and solve many simple problems themselves from their computers. The e-commerce debugger is analogous to a programming language debugger used in software development. It allows the user to systematically investigate the possible causes of an error by examining processes and data at varying levels of detail until the problem is discovered.

Debugging is detective work. It requires generating hypotheses, and testing them by a process of elimination. Of course, software debuggers are used by expert programmers and have interfaces designed to be used by expert users. For an e-commerce debugger to be effective, it must have a user interface designed for use by non-expert users, with simple interfaces and ample explanation and visualization facilities. Of course we don't expect end-users to understand and use a software debugger like programmers do. However, we do believe that experienced computer users have enough understanding about cause-and-effect relationships among entities they're familiar with to be able to use the tool we will present, and they will effective diagnosing the problems they face.

Expert programmers are confident that no matter where a bug might be, systematic investigation will almost always find the problem if a fine enough level of detail is examined. That gives them confidence in using the technology. We intend to instill that level of confidence in users who participate in e-commerce transactions. When the user sees some data in a page that looks wrong, she is able to mark it as wrong and the tool guides her through the process of isolating the problem further until a particular process or data is found to be at fault. At that point the user receives a record of her process of diagnosis that can be shared, with a customer service representative, for instance.

In the past, it would have been impossible to provide practical e-commerce debugging tools because the raw material that such tools would need - information about transactions, user identification, vendor specifications, payments, and so on, would exist only on paper. With higher-level annotations of web pages enabled by the advent of the Semantic Web, however, we expect that it will be practical to capture that information via web browsers, and provide debugging facilities operating through the browsers.

1.5.1 Web Process Models

Web processes differ in an important way from software, however. While source code describes the behavior of just software, web processes require describing the actions of users and web sites. Models of a user's actions and the reactions of the web site must somehow be provided. As we will see, the e-commerce debugger is agnostic about the source of these process models. It supports customized models particular to a web site, or generic model for an action, such as making a purchase, that is similar on multiple web sites. They may be provided by the web site, vendor or even the user herself through training.

1.6 End-User Debugging

We actually see this work in the context of a broader goal of developing end-user debugging tools for all sorts of computer interactions, not just for e-commerce transactions. Computer users lose time and become frustrated in dealing with the intricacies of software installation, system maintenance, and problem solving specific applications. Why can't the computer itself know what it is doing and help us solve the problems we are having with it?

Why can't we ask a computer in any situation, "What are you doing right now?", "Why are you doing that?", "What does this error message mean?", and other questions that you might reasonably ask a knowledgeable human assistant? If something goes wrong, why can't we trace the problem back through the steps that led up to it? Why can't we formulate hypotheses about what might have been wrong, and test them until we get to the answer?

Many of the cognitive processes involved in diagnosis and repair of e-commerce situations have analogies in many kinds of general system interaction as well. So we think our results will generalize to many other areas of computer interaction.

But e-commerce is a good place to start, because the transactions themselves are procedurally very simple, compared to the operation of an operating system or specialized application. People understand the basic concepts of financial transactions, and since everyone participates in such transactions, interfaces to deal with them will find a wide audience.

The economic value of improving the situation is enormous, and vendors have not so far tackled the problem in a comprehensive way. Because many actions involve multiple parties, such as an online purchase involving the vendor, the consumer's credit card company and the shipping service, consumers need help from tools that operate on their behalf, rather than on behalf any particular vendor.

Further, we think the interfaces developed to support end-users in debugging could also influence future debugging tools for professional software developers. Since debugging accounts for approximately half of all software development costs, the potential economic payback is enormous here, too. Because development of debugging tools has so far been limited to expert professionals, we believe that insufficient attention has been paid to the usability of such tools, and we believe that a focus on supporting the needs of non-programmers could benefit in these situations as well.

Chapter 2

Introduction to Woodstein with Examples

2.1 Introduction to Woodstein

Woodstein is a software agent that works with a user's web browser to answer questions like "How did that data get that value?" "Why did that happen" and "What's happening now?". It monitors a user's actions on the web, such as browsing an online retailer and adding items to a shopping cart, to create a record of the user's overall process, in this case, making a purchase. It is then able to answer questions about the history and current status of the process, as well as how data in the process was set.

Woodstein matches a user's actions to the steps of an abstract model for the process. Through this process of recognition, it knows to look for more information about the process on other web pages and web sites, even if user never visited them. By seeing the user select a credit card and shipper for a purchase, Woodstein knows to go to the sites of the bank and shipper for more information about the status of the purchase, including whether it has been paid for and delivered.

Later, when the user is looking at other pages with data about the process, such as the credit card transactions history page, the charge itself can be inspected. The history of the purchase process can be retrieved and reviewed, making it convenient to understand the context of the data, the charge.

Woodstein explains the history of the data and actions of a process through several different views. As a user is performing a step of an action, it can show where that step fits into the overall process. Another view shows the history of a process and allows the user to revisit any point. When inspecting a data item, a view displays an automatically generated audit trail it, enabling the user to jump among pages in which it appears. The user can jump from the charge amount in the transactions page to a saved copy of the order confirmation page in which the amount appears.

In addition to providing views on the history of processes and data, Woodstein also provides help in diagnosing problems when something goes wrong. When a user notices that something doesn't "look right", simply clicking on a menu item tells the agent. Woodstein responds by identifying processes or data that could have contributed to the error or unexpected result. Through this assistance, the user is able to identify the exact step or data that either caused an error, or created an unexpected result.

Woodstein's features can best be understood through examples. Through the rest of this chapter, we will see several examples each showing some of Woodstein's features. We will see how Woodstein:

tracks user actions and recognizes user goals saving the users interactions and retrieving information to create an entire history of the action

provides help and justification for the user current action at varying levels of detail and specificity

supports inspection of information in pages when the user want to know more about data and processes included within them

explains the history of processes and data through easily-understood views that visualize their relationships

shows all pages related to a user action including saved copies of exactly the pages the user interacted with and related pages that Woodstein retrieves that include more information

guides the user in diagnosing problems and prepares a record of the diagnosis

Problems arise because of mistakes by either the user or the web site, or because the user has an incorrect mental model of the process. It is important that the agent provides help in all cases, because when a problem symptom is found, it is not known what the source of the problem is or whether there even is a problem. In this chapter we will see examples from each category.

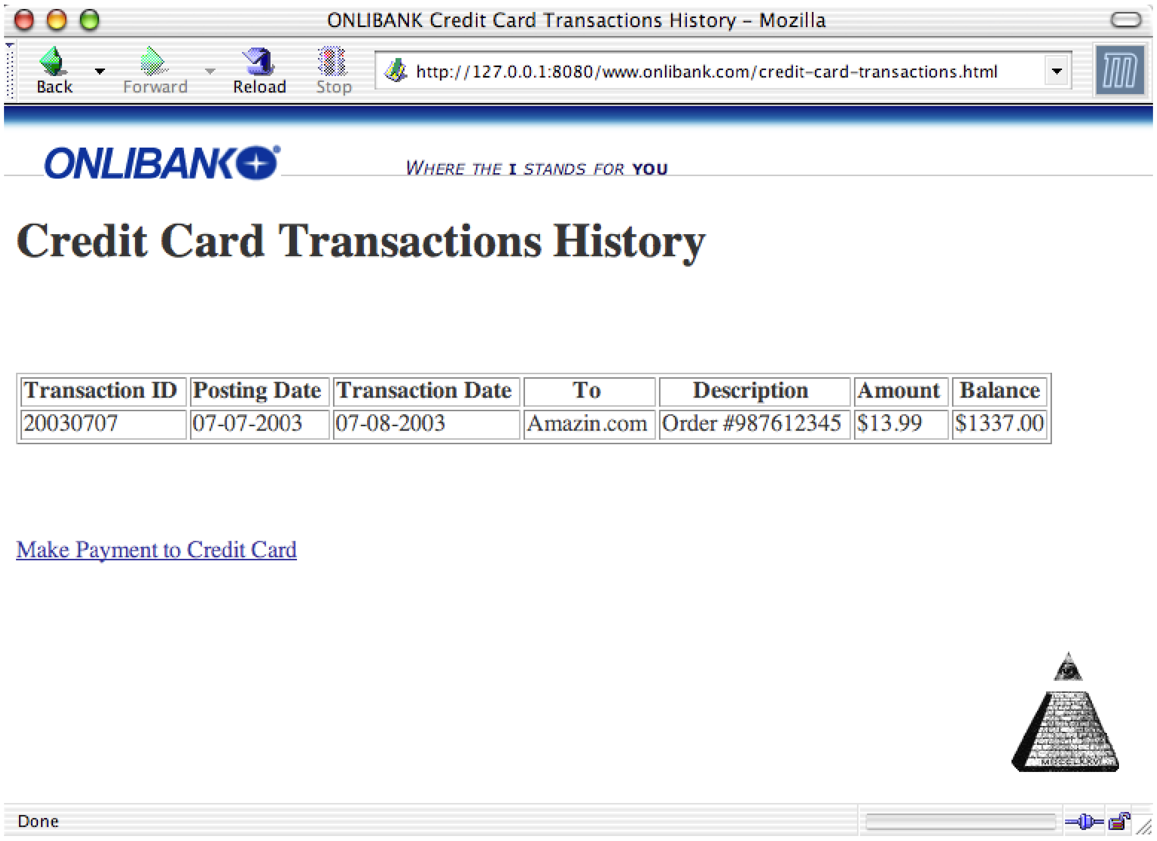
2.2 Purchase Scenario: Inspecting a Credit Card Transaction with Woodstein

In this first scenario involving a purchase, we will see some of Woodstein's features and how it manages a user's actions and data and supports inspecting those actions and data in pages. We will see its views presenting process and data histories, as well as saved pages that either the user interacted with or were retrieved by Woodstein. Future examples will make use of these features as well as other, more advanced features, but this example will provide a glimpse of what Woodstein does and how it works.

Anyone who has a credit card has had the experience of looking at an unfamiliar charge and trying to remember what it was for. Often the description is unclear or too terse, or perhaps the vendor is unknown. In situations like these, the information gathered by Woodstein becomes useful. By tracking the user's original steps in making the purchase, the agent is able to connect those steps with the charge that appears on the credit card. It presents the record it compiled at moments like this one - when the user finds something unusual.

The user looks at the unfamiliar charge in Figure 2-1, but doesn't remember anything about it. The vendor is unknown - it looks like Amazon.com, but it isn't - and the transaction description isn't helpful in reminding him. He is frustrated and wants to know where it came from, so he decides to consult Woodstein. Clicking on Woodstein's logo turns on Woodstein's inspection mode, revealing all of the data and processes it found in the page. In this case, he wants to find out the origin of this charge and how it was created. Each individual data item of the transaction can be interacted with separately, or the entire transaction can be inspected by selecting its name, the transaction ID. Pressing the mouse button down on the transaction ID button causes a context-sensitive menu to appear with questions the user can ask (Figure 2-2). The user wants to know why the transaction was created and selects the item "Why was this created?".

Letting go of the mouse button opens a view in a pop-up window (Figure 2-3).



ONLIBANK Credit Card Transactions History - Mozilla

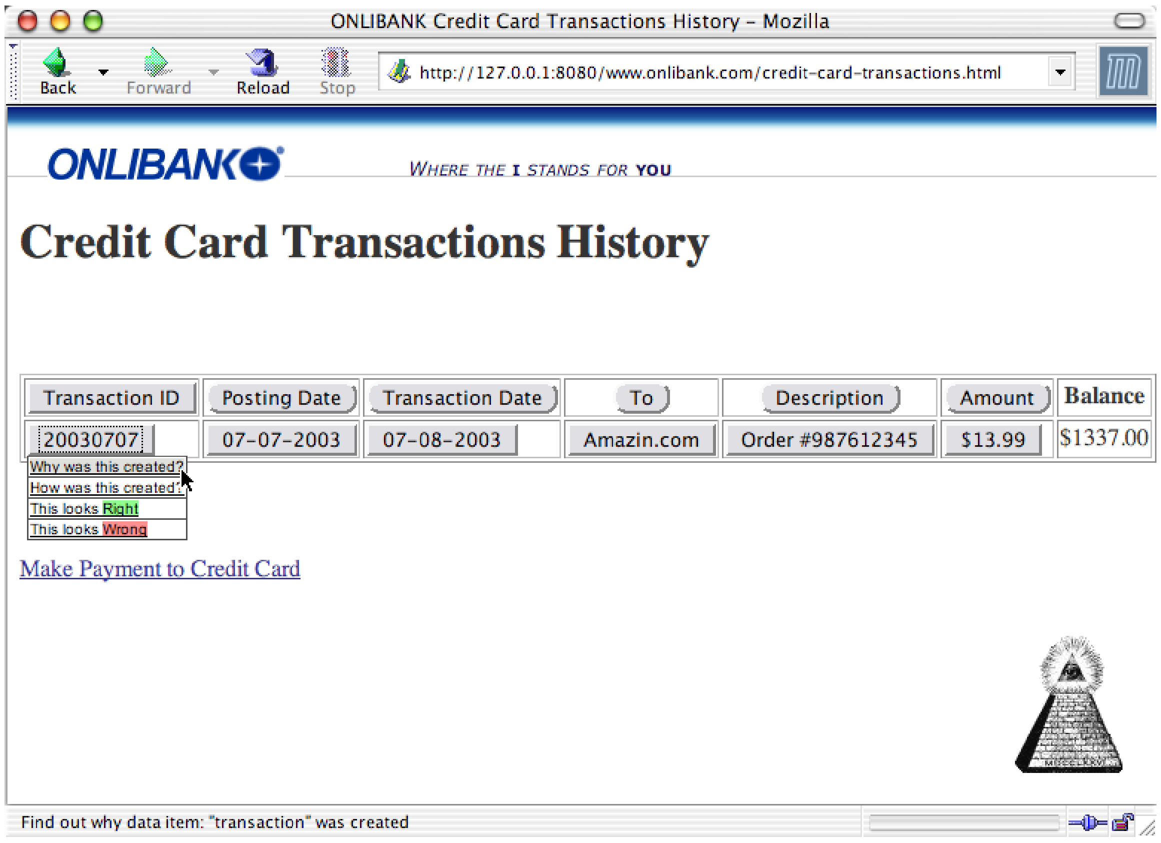
http://127.0.0.1:8080/www.onlibank.com/credit-card-transactions.html

Credit Card Transactions History

Transaction ID	Posting Date	Transaction Date	To	Description	Amount	Balance
20030707	07-07-2003	07-08-2003	Amazin.com	Order #987612345	$13.99	$1337.00

Make Payment to Credit Card

Figure 2-1: Looking at an unfamiliar transaction



ONLIBANK Credit Card Transactions History – Mozilla

http://127.0.0.1:8080/www.onlibank.com/credit-card-transactions.html

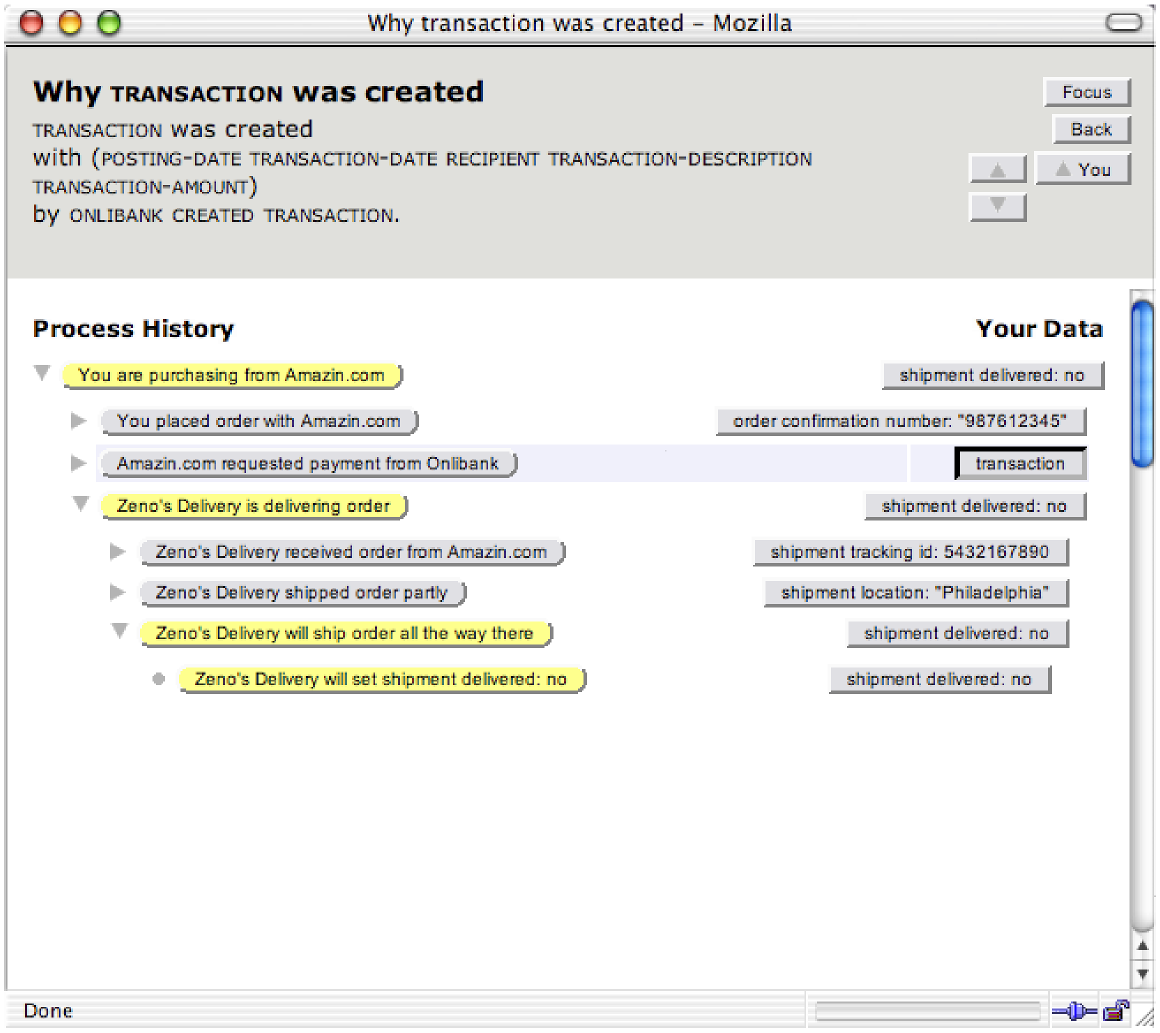
Credit Card Transactions History

Transaction ID	Posting Date	Transaction Date	To	Description	Amount	Balance
20030707	07-07-2003	07-08-2003	Amazin.com	Order #987612345	$13.99	$1337.00

Make Payment to Credit Card

Find out why data item: "transaction" was created

Figure 2-2: Asking why an unfamiliar transaction was created



Why TRANSACTION was created

TRANSACTION was created
with (POSTING-DATE TRANSACTION-DATE RECIPIENT TRANSACTION-DESCRIPTION TRANSACTION-AMOUNT)
by ONLIBANK CREATED TRANSACTION.

Process History	Your Data
You are purchasing from Amazin.com	shipment delivered: no
You placed order with Amazin.com	order confirmation number: "987612345"
Amazin.com requested payment from Onlibank	transaction
Zeno's Delivery is delivering order	shipment delivered: no
Zeno's Delivery received order from Amazin.com	shipment tracking id: 5432167890
Zeno's Delivery shipped order partly	shipment location: "Philadelphia"
Zeno's Delivery will ship order all the way there	shipment delivered: no
Zeno's Delivery will set shipment delivered: no	shipment delivered: no

Figure 2-3: Viewing why the unfamiliar transaction was created

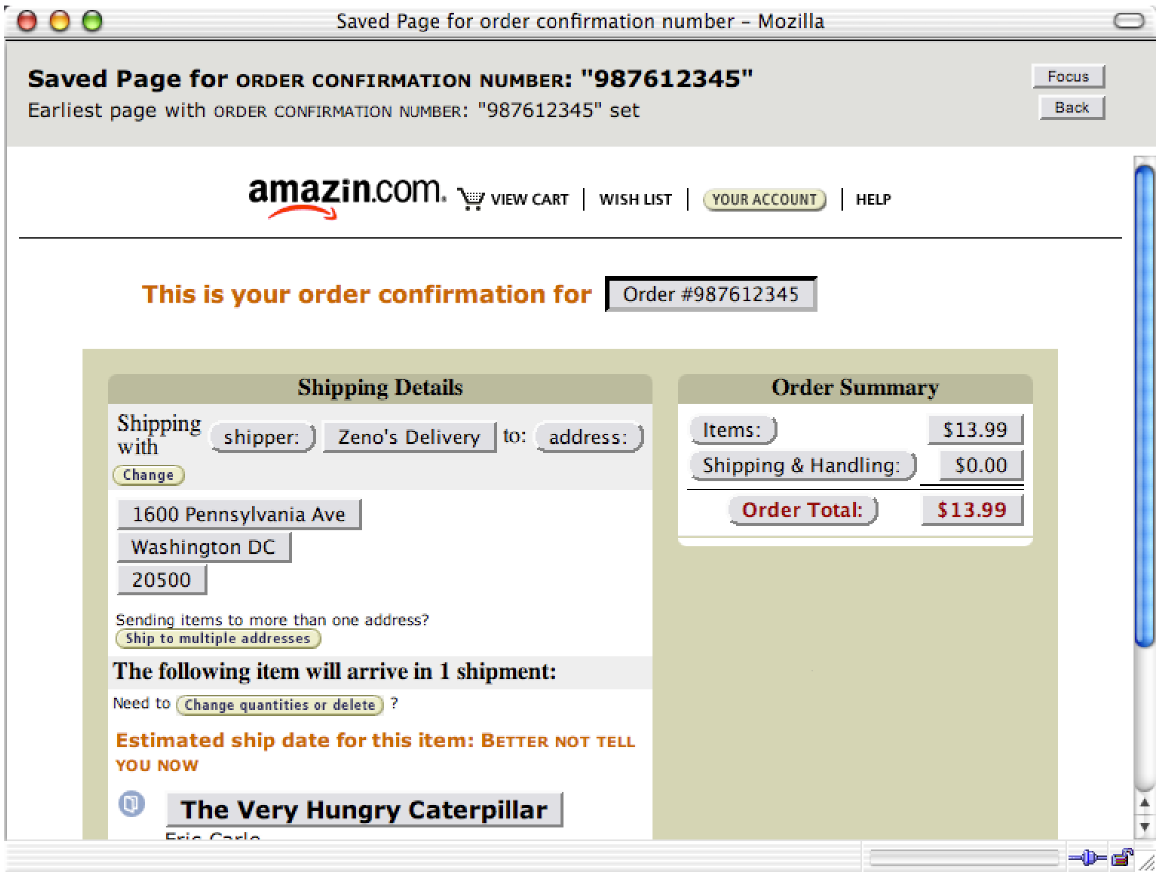
The new window shows the overall history of the user's purchase with English descriptions of the process steps. Woodstein created this record by matching the user's original concrete steps in placing the order with its abstract process model describing how purchases happen. While placing the order, the user selected a credit card and shipper and based on these details, Woodstein knew to look at their web sites, match the transaction and find out about the status for the transaction and delivery. In this view, Woodstein presents the information it gathered from the three sites affected by the purchase.

In the top frame of the window, in grey, the view explains why the credit card transaction was created. It was created with all of the individual data shown in parenthesis, including the "posting date", "transaction date", "recipient" and so on. All data is created by a process and the process that created the transaction was "Onlibank created credit card transaction".

In the bottom frame of the window, Woodstein shows the hierarchical structure of the process. It starts out with the overall goal, "You are purchasing from Amazin.com". Below that, the purchase action is broken down into three steps, "You placed order with Amazin.com", "Amazin.com requested payment from Onlibank" and "Zeno's Delivery is delivering order". The descriptions of these steps indicate that the first two steps have completed since they are in the past tense. Each completed step has the data it produced as its result. The user's order action produced an order confirmation number, while the vendor's payment request produced the credit card transaction that the user is currently inspecting. The button for that data looks pressed in because that it is the data that is currently selected and being inspected. Each step also has a small icon on its left side that indicates the step has more sub-steps that are hidden.

Below the two completed steps of the purchase is the last step. Its description, "Zeno's Delivery is delivering order" is in the present tense and it is marked yellow, indicating that it is still ongoing. Woodstein also opened revealed it sub-steps so the user can easily the steps that remain. Since the entire purchase itself is still ongoing, it too in the present tense and marked yellow and its sub-steps were automatically revealed.

This window is the "why" process history view. It is opened by Woodstein in response to the user asking "why" some action happened, or some data was set. It presents the process in a hierarchical format, to show that, that each step takes place because of the large process it is a part of. In this case, the shipper set the location of the order because


Saved Page for order confirmation number – Mozilla

Saved Page for ORDER CONFIRMATION NUMBER: "987612345"
Earliest page with ORDER CONFIRMATION NUMBER: "987612345" set

Focus
Back

amazin.com. VIEW CART | WISH LIST | YOUR ACCOUNT | HELP

This is your order confirmation for Order #987612345

Shipping Details

Shipping with shipper: Zeno's Delivery to: address:
Change

1600 Pennsylvania Ave
Washington DC
20500

Sending items to more than one address?
Ship to multiple addresses

The following item will arrive in 1 shipment:

Need to Change quantities or delete ?

Estimated ship date for this item: BETTER NOT TELL YOU NOW

The Very Hungry Caterpillar

Order Summary

Items:	$13.99
Shipping & Handling:	$0.00
Order Total:	$13.99

Figure 2-4: Viewing the saved confirmation page for the order

it shipped the order to an intermediate location. It shipped the order to an intermediate location because it is the process of delivering the order. It is in the process of delivering the order because the user initiated a purchase. This explanation is cumbersome to read as text, but it is easily comprehensible in an interactive and inspectable diagram.

At this point, the user still wants to know where this charge originated and what the item was that it paid for. The transaction itself is the currently selected button, but now he wants to inspect the order confirmation, the result of “You placed order with Amazin.com”, and he clicks that button. A new pop-up window, with the saved order confirmation appears (Figure 2-4).

While the user was placing the order, Woodstein saved a copy of this confirmation and is able to show it now. The user sees the item he bought and now knows what the transaction on his credit card was for.

2.3 How Woodstein Presents Data and Processes

So far, we've seen a simple example that shows some Woodstein's interface for presenting the history of user processes. At this point it will be helpful to look in more detail at how Woodstein manages and presents data and processes in order to understand what its views show.

As we've seen, Woodstein tracks data and processes:

Data Items make up much of the information in web pages. They appear in web pages in two forms:

Simple Data Items include prices, dates, and IDs, as well as whether something happened or whether a requirement was met.

Composite Data Items are records that consist of other data items. An example is the user's credit card transaction that consists of a posting date, transaction date, recipient as well as other data items. Composite data items can be inspected via their identifiers, which act as names for the entire record. The transaction ID and order confirmation number are examples of identifiers.

Processes include specific actions like the user clicking a button or loading a page, as well as a web-site shipping an order. They can also be more abstract, such as the entire process of the user making a purchase. They are always performed by some entity, either the user or a web site. There are several ways in which processes appear in web pages:

Form Controls include buttons the user clicked, text areas in which text was typed and so on. In this case the corresponding process is the user's original interaction with the control.

Names of Data Items in which the process is the setting of the data item. In the confirmation page, the two buttons appear for the order total, one for the label "Order Total:" and one for the amount itself, "$13.99". The amount corresponds to the order total data item, and label corresponds to the process in which the data item was set.

Instructions or Policies for a web site. These include instructions explaining to the user how to do something, or a description of how a web site performed some action.

All of these will become more clear as we continue with the example.

Each data item is set as the result of a process. Processes take data items as inputs and set a data item result.

Woodstein keeps track of data and processes similarly, so we refer to them as representations or "reps". It is important to note that Woodstein isn't involved in what the data represents; it doesn't know any more about the actual ordered item than what's shown on the web sites' pages. Nor does it play any role in how the data and processes appear in pages while the user browses. Woodstein only keeps track of data and processes when they do appear in web pages. Reps are these records that Woodstein keeps for data and processes. In tracking reps, Woodstein saves the pages in which they first appear. When the user inspects the transaction, he interacts and views Woodstein's rep for it. Since Woodstein presents both data and processes as buttons, we will also refer to the buttons themselves as reps.

There are some final things to keep in mind when interacting with Woodstein.

- Buttons for data items are rectangular, buttons for processes are rounded.
- Pressing the mouse button down on a rep shows its menu of options. Letting go of the button while over an option selects that option.
- Either clicking on a rep or using its menu to interact with it causes it to be the "selected rep" that becomes the focus of all open views. At any given moment there is at most one selected rep. Its button appears pressed-in, while all other buttons appear popped-out.
- With multiple views it can be tough to keep track of what corresponds to what, so moving the mouse into a rep in one view highlights the same rep in other views.
- The saved page view always shows Woodstein's saved page for the selected rep. This is the page in which the process is described, or in which the data item was first set, or it is an empty page if Woodstein had to infer that the process happened or data

item was set. Since the first page in which the order confirmation number appeared is the order confirmation page, that is its saved page.

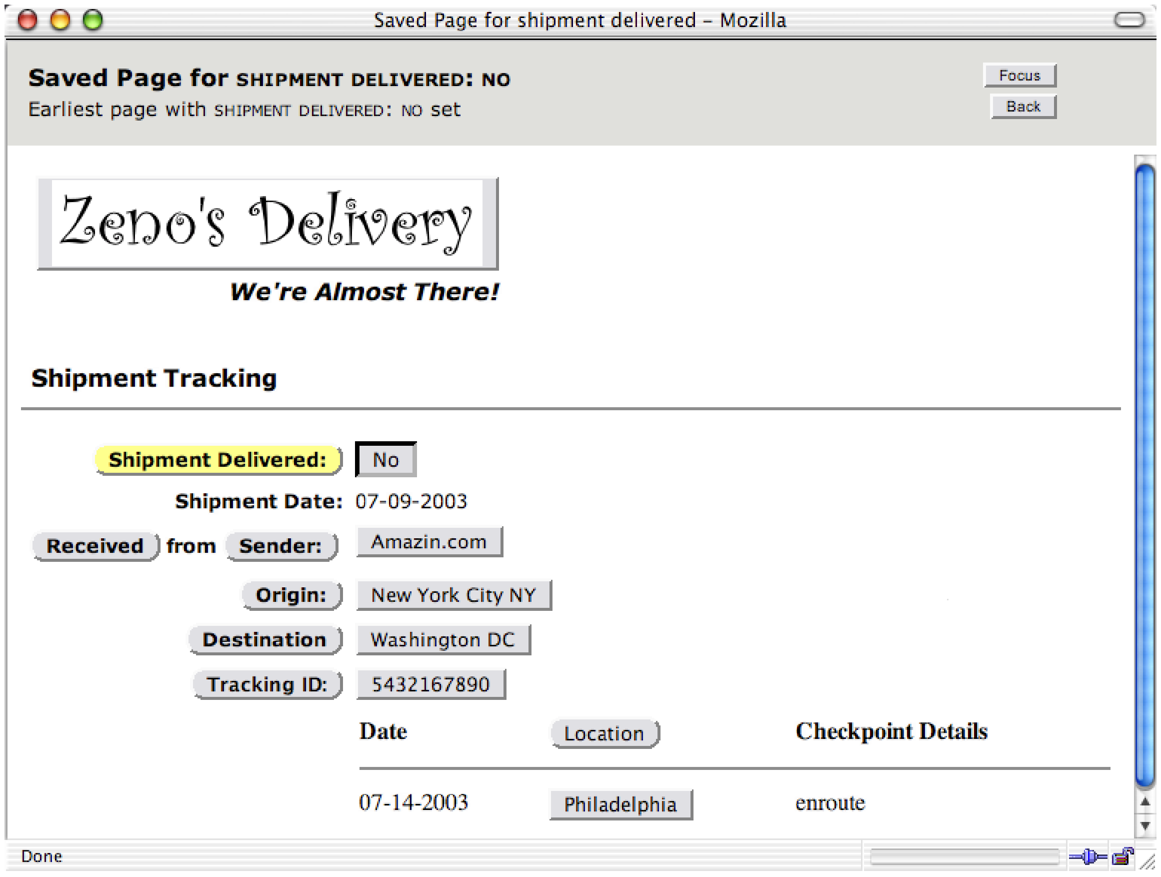
2.4 Learning about a Purchase with Woodstein

Having discovered what the original charge was for, the user wants to find out what happened to the ordered book. Where is it? Why hasn't it been delivered yet? What's taking so long? Though some sites like Amazon.com feature order tracking, it is inconvenient to go to the site, sign-in to access the account history, look at the recently shipped items and figure out which corresponded to this credit card charge. For multiple charges on many sites, doing this matching can be a burden. The problem is that although a site like Amazon.com is able to integrate with the shipper's site and present the purchase status and shipping status together, it does not integrate with the bank's site and allow the user to access the history of the purchase through the credit card charge. Even worse, most sites offer far less help than Amazon.com and don't even try to show the shipping status, leaving it up to the user to integrate the information provided by the sites of the vendor, the shipper and the bank in order to understand what is going on.

By tracking the user's order as he was placing it, and collecting related information, Woodstein is able to show the purchase status. More importantly, it has a record of the history of the purchase including the original pages the user interacted with, such as the order confirmation, as we saw.

At this point the user wants to know about the shipment delivery status for the book. Looking at the purchase process in the process history view, he sees its current status that the shipment hasn't been delivered and, looking inside the shipping action, he sees that its location is Philadelphia. He clicks on the status, and Woodstein shows a saved copy of the current tracking page for the shipment (Figure 2-5). Although it is traveling slowly, it looks like the shipment is on it's way and will be delivered at some point in the future.

Now he's curious about this shipping company. How was it selected to deliver his book? What made him choose this company? Since Woodstein saw the original purchase, it recognized the name of the shipper and allows it too to be inspected. He presses the mouse button down on the button for the shipper's logo and asks "How was this set?" (Figure 2-6).


Saved Page for shipment delivered – Mozilla

Saved Page for SHIPMENT DELIVERED: NO
Earliest page with SHIPMENT DELIVERED: NO set

Zeno's Delivery
We're Almost There!

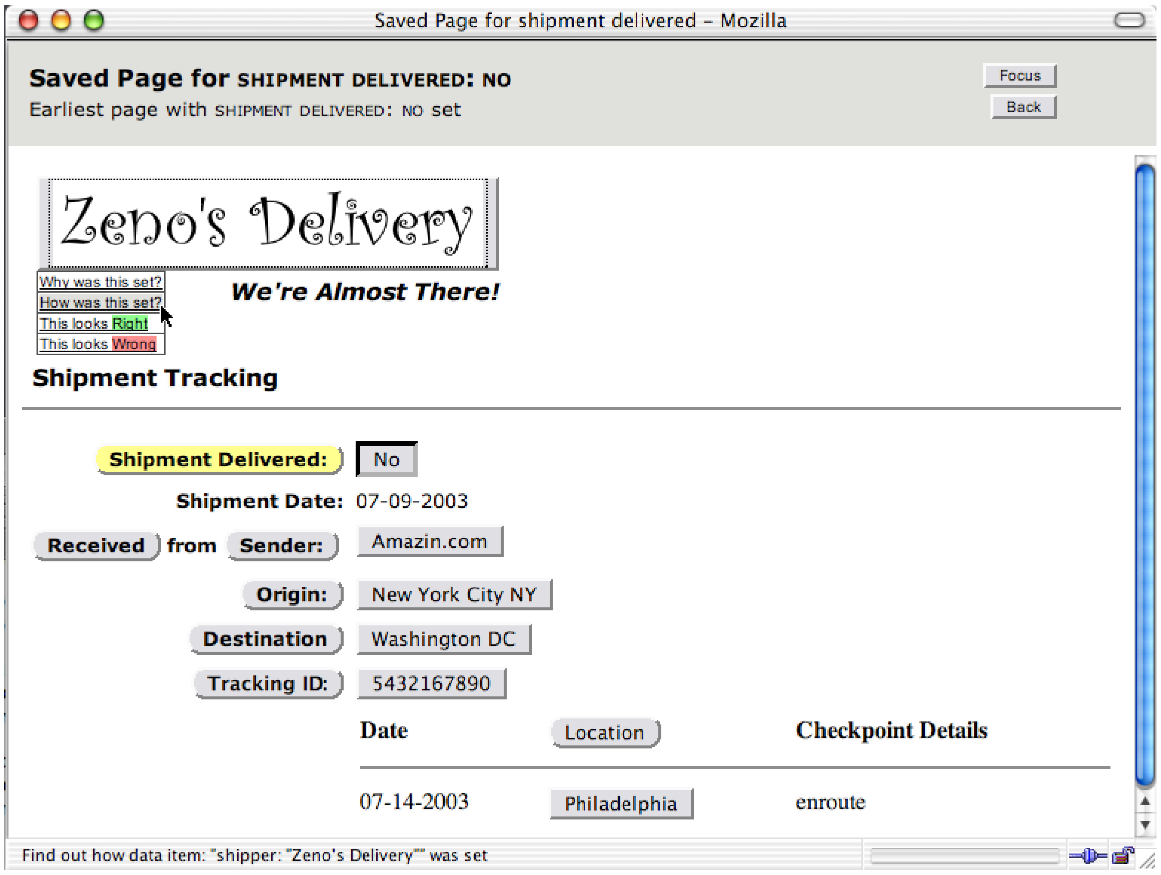
Shipment Tracking

Shipment Delivered:	No
Shipment Date:	07-09-2003
Received from Sender:	Amazin.com
Origin:	New York City NY
Destination	Washington DC
Tracking ID:	5432167890

Date	Location	Checkpoint Details
07-14-2003	Philadelphia	enroute

Done

Figure 2-5: Viewing the saved page for the shipment delivery status


Saved Page for shipment delivered – Mozilla

Saved Page for SHIPMENT DELIVERED: NO

Earliest page with SHIPMENT DELIVERED: NO set

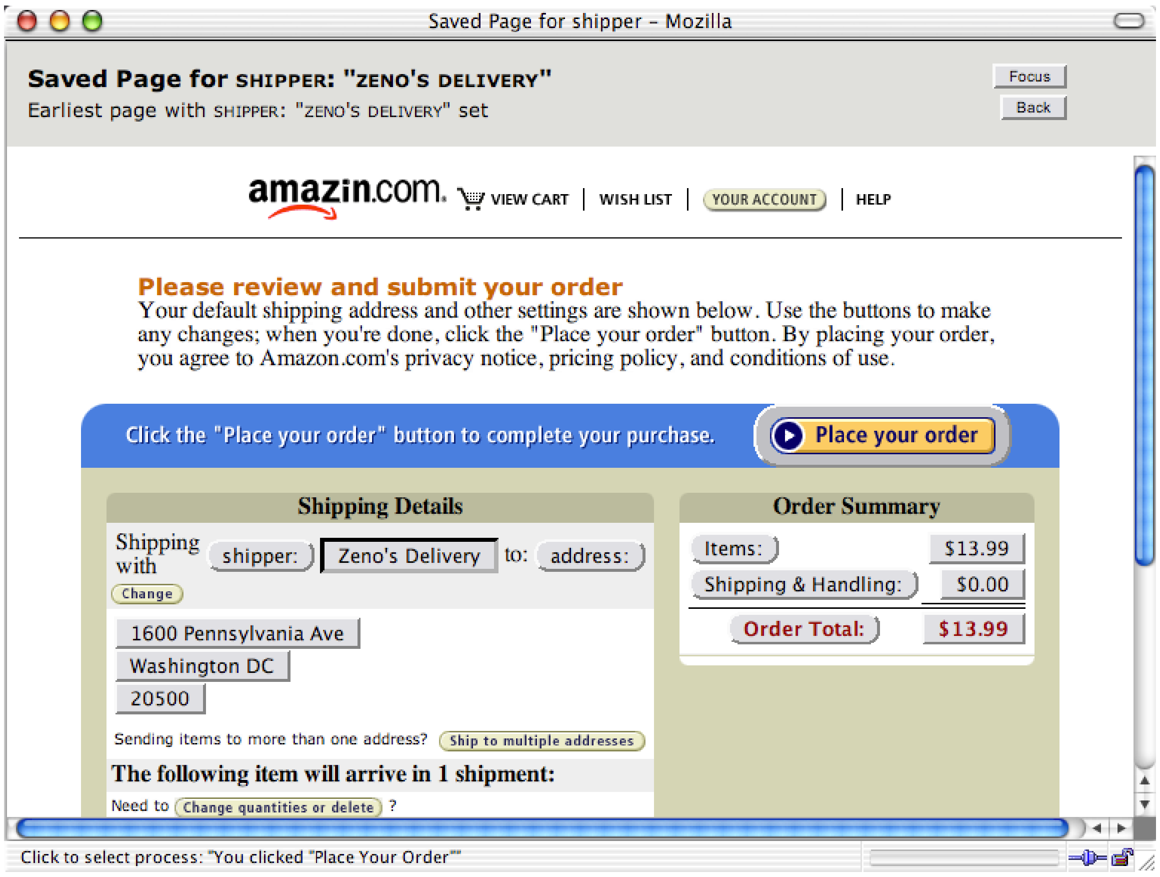
Shipment Tracking

Shipment Delivered:	No
Shipment Date:	07-09-2003
Received from Sender:	Amazin.com
Origin:	New York City NY
Destination	Washington DC
Tracking ID:	5432167890

Date	Location	Checkpoint Details
07-14-2003	Philadelphia	enroute

Find out how data item: "shipper: "Zeno's Delivery"" was set

Figure 2-6: Asking how the shipper was set


Saved Page for shipper - Mozilla

Saved Page for SHIPPER: "ZENO'S DELIVERY"
Earliest page with SHIPPER: "ZENO'S DELIVERY" set

Please review and submit your order
Your default shipping address and other settings are shown below. Use the buttons to make any changes; when you're done, click the "Place your order" button. By placing your order, you agree to Amazon.com's privacy notice, pricing policy, and conditions of use.

Click the "Place your order" button to complete your purchase.

Shipping Details
Shipping with shipper: Zeno's Delivery to: address:
1600 Pennsylvania Ave
Washington DC
20500
Sending items to more than one address?
The following item will arrive in 1 shipment:
Need to Change quantities or delete ?

Order Summary

Items:	$13.99
Shipping & Handling:	$0.00
Order Total:	$13.99

Click to select process: "You clicked "Place Your Order""

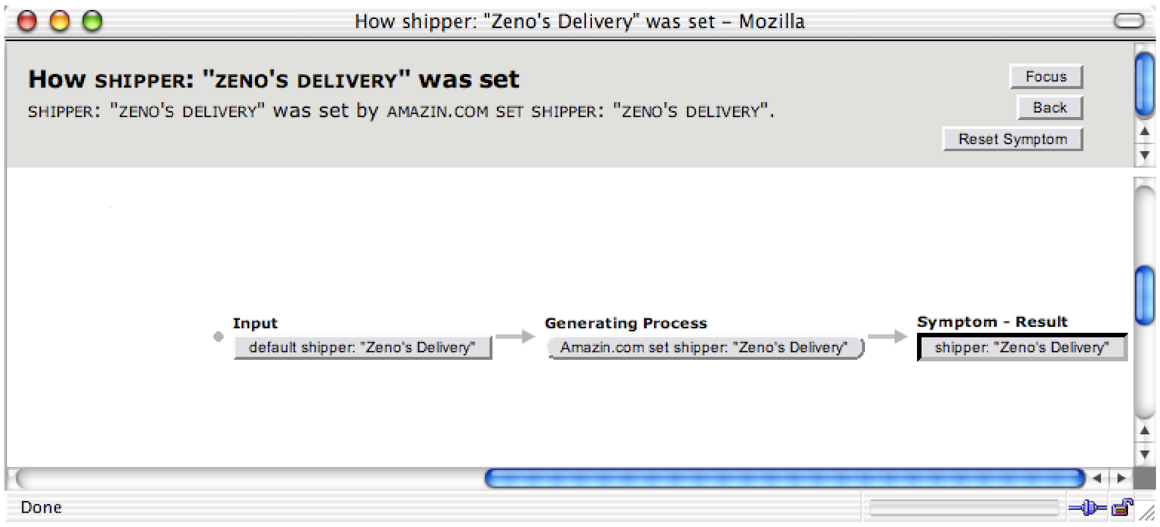
Figure 2-7: Viewing the saved page for the shipper

Woodstein updates the saved page view to show the page in which it verified that Amazin.com set the shipper for the order (Figure 2-7).

In addition to updating the saved page view, Woodstein created a new pop-up window with a new view to explain how the shipper was set (Figure 2-8).

Like the "why" process history view, this "how" data history view features a grey frame at the top that explains how the "shipper" data item was set to have the value of "Zeno's Delivery". It resulted from "Amazin.com set shipper". Below that frame is a display with the symptom/result that the user was inspecting, the "shipper" data all the way on the right. Before that, in the middle, is the process that generated it, "Amazin.com set shipper". Finally, all the way on the left is the input to that process, the "default shipper". It appears that the Amazin.com just set the shipper to be the default shipper.

Unlike the "why" view, which shows the overall history of the process, the "how" view displays an audit trail showing the history of the data of the process. When inspecting a data item, the user cares only about the steps involved in computing its value. In this case,


How shipper: "Zeno's Delivery" was set – Mozilla
Focus
Back
Reset Symptom
Done

How SHIPPER: "ZENO'S DELIVERY" was set

SHIPPER: "ZENO'S DELIVERY" was set by AMAZIN.COM SET SHIPPER: "ZENO'S DELIVERY".

Input
default shipper: "Zeno's Delivery"
Generating Process
Amazin.com set shipper: "Zeno's Delivery"
Symptom - Result
shipper: "Zeno's Delivery"

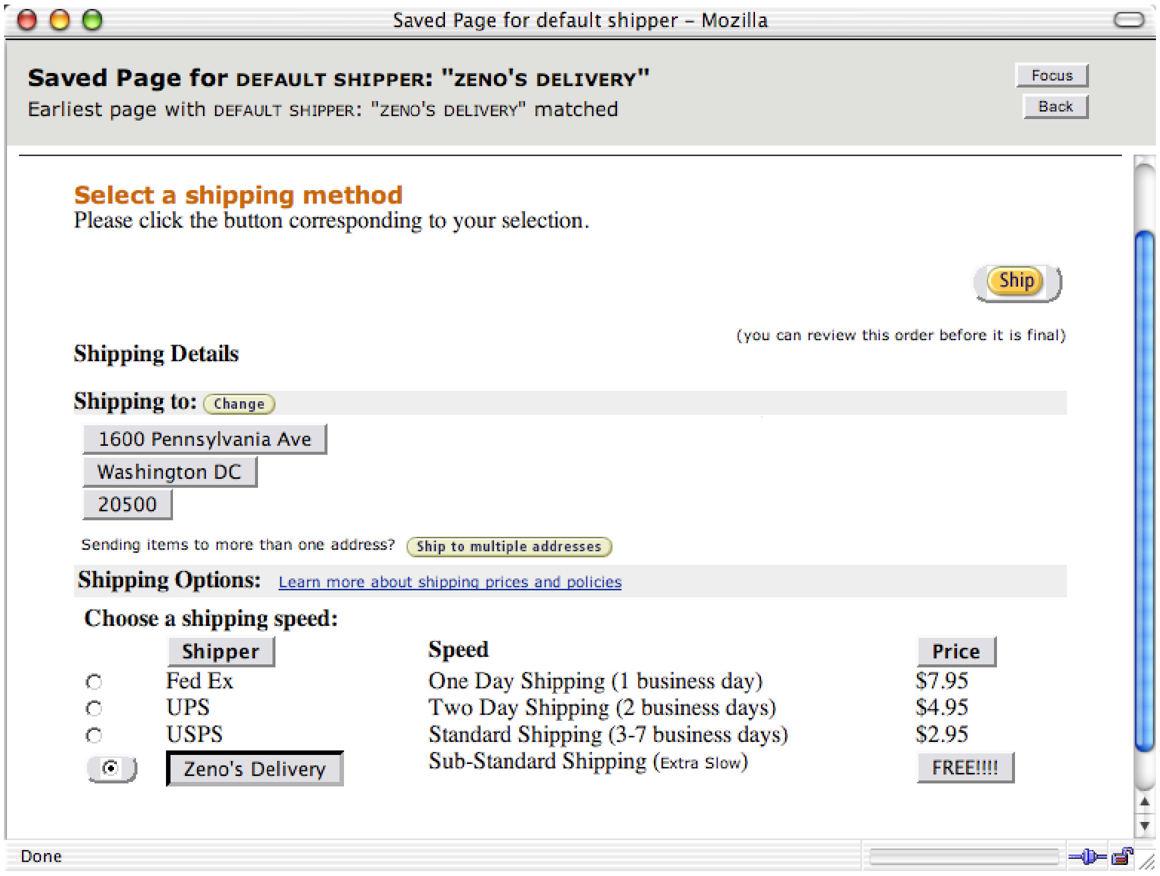
Figure 2-8: Viewing how the shipper was set

the user doesn't care about how all of the different data items for the order were set, he just cares about the subset of the process in which the shipper was set. This view allows him to easily jump to the page in which it was set. More generally, the view allows the user to traverse back through the process from the perspective of the data. Similar views in program debuggers are referred to as program slices.

The user wants to know how the default shipper became the shipper for the order, so he clicks on the "default shipper" rep and Woodstein updates the saved page view (Figure 2-9).

At the bottom, he sees the button that had been set, the name of the shipper that was selected, and the price for shipping. He also sees that this is the extra slow, sub-standard shipping option and wonders why he ever selected this shipper. He clicks on the "shipper" rep again and looks over to the process history view that Woodstein has updated to focus on it (Figure 2-10).

Woodstein expanded both the "You placed order with Amazin.com" step of the purchase and, below that, the "You selected shipping method" step. He sees that the step of Amazin.com setting the shipper happened as part of his selecting the shipping method and looks at the steps before it: "You saw shipping method selection page", "You left default shipper selected", "You clicked ship". It looks like he did indeed just leave the default method selected before continuing with his order, and that's how this shipper was selected. Having exercised his brain enough with solving that mystery, the user decides to do some mid-afternoon jogging.



Saved Page for default shipper – Mozilla

Saved Page for DEFAULT SHIPPER: "ZENO'S DELIVERY"
Earliest page with DEFAULT SHIPPER: "ZENO'S DELIVERY" matched

Select a shipping method
Please click the button corresponding to your selection.

(you can review this order before it is final)

Shipping Details

Shipping to:

1600 Pennsylvania Ave
Washington DC
20500

Sending items to more than one address?

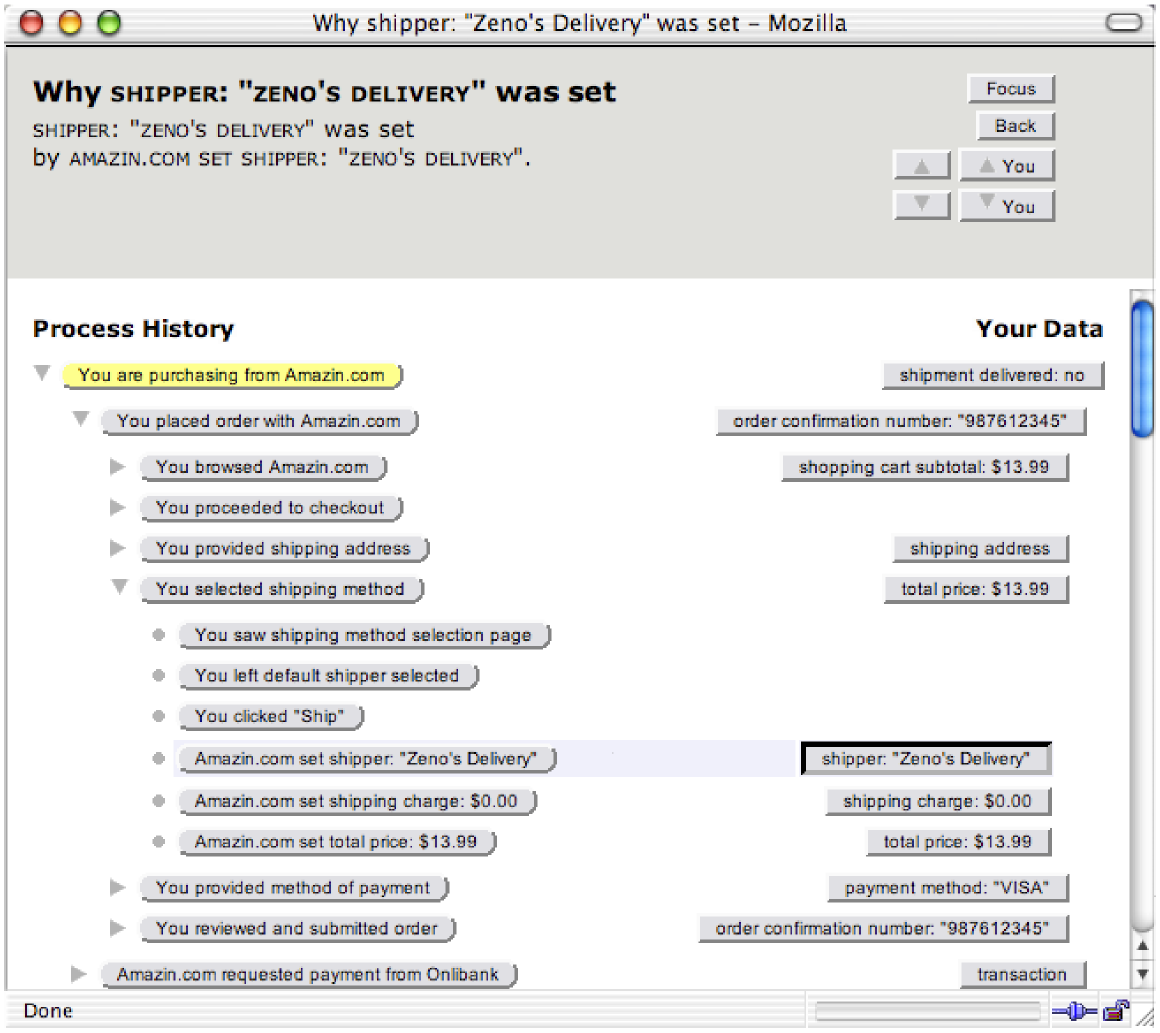
Shipping Options: Learn more about shipping prices and policies

Choose a shipping speed:

	Shipper	Speed	Price
○	Fed Ex	One Day Shipping (1 business day)	$7.95
○	UPS	Two Day Shipping (2 business days)	$4.95
○	USPS	Standard Shipping (3-7 business days)	$2.95
[✓ ●]	Zeno's Delivery	Sub-Standard Shipping (Extra Slow)	FREE!!!!

Done

Figure 2-9: Viewing the saved page for the default shipper


Why shipper: "Zeno's Delivery" was set – Mozilla

Why SHIPPER: "ZENO'S DELIVERY" was set

SHIPPER: "ZENO'S DELIVERY" was set
by AMAZIN.COM SET SHIPPER: "ZENO'S DELIVERY".

Process History **Your Data**

You are purchasing from Amazin.com	shipment delivered: no
You placed order with Amazin.com	order confirmation number: "987612345"
You browsed Amazin.com	shopping cart subtotal: $13.99
You proceeded to checkout	
You provided shipping address	shipping address
You selected shipping method	total price: $13.99
You saw shipping method selection page	
You left default shipper selected	
You clicked "Ship"	
Amazin.com set shipper: "Zeno's Delivery"	shipper: "Zeno's Delivery"
Amazin.com set shipping charge: $0.00	shipping charge: $0.00
Amazin.com set total price: $13.99	total price: $13.99
You provided method of payment	payment method: "VISA"
You reviewed and submitted order	order confirmation number: "987612345"
Amazin.com requested payment from Onlibank	transaction

Done

Figure 2-10: Viewing why the shipper was set

2.4.1 Summary

This example was one in which the user made an error and was able see how his actions caused the error. We saw how the user is able to inspect processes and data in pages to learn more about them. We saw that with the "why" view, the user was able see the overall history of a process, as well as where individual steps fit in. We saw that with the saved page view, the user is able to see pages that he interacted with and pages where processes and data first appear, even if he never saw them. Finally, we briefly saw the user use the data history view to see the history of data in the process.

2.5 How Woodstein Manages and Presents Views

Through the first example, Woodstein opened new views and automatically updated existing views in response to the user's inspection of reps. It will be helpful to discuss how it does that.

As we saw earlier, at any given moment while the user is inspecting, there is a single selected rep. This rep is selected not only in the view the user selected it in, but in all other views. When a user selects a rep by clicking on it or asking about it, all views update to explain it. The "why" process history view shows its context in the overall history of the process and the result of each step, the "how" data history view shows the processes and data used to compute it, and the saved page view shows the page in which it first appears.

2.5.1 Woodstein Shows Only Relevant Data Items and Processes

Woodstein distinguishes between why a rep happened or was set, from how it happened or was set. The first focuses on the history of the overall process. The reason why each step below the top action in a process happens is because of the process above it that it is a part of. For instance, the user places an order because he is making a purchase. In programming, this hierarchical structure of processes is known as the "control-flow".

How a rep happens or is set can also be understood by seeing the history of the data in the process known as the "data-flow" of the process. A data item may result from a process that generates it, as the shipper resulted from Amazin.com setting it. Or Woodstein may find a data item in a page, but have no process description for it, as was the case with the default shipper. Woodstein first learned about this default value when it appeared in the shipping method selection page loaded by the user in originally placing the order. Woodstein has no record of the history for these data items.

Rather than overload the user with the entire history of all data and steps of a process, the process history view and data history view update differently depending on the selected rep. The process history view always shows the user's overall action, such as making a purchase, and temporarily opens enough steps to show the context of the selected rep. Clicking on the icon on the side opens and closes the steps, showing and hiding their sub-steps, respectively. When a new rep is selected, the process history view closes the steps it temporarily opened for the old selected rep and temporarily opens steps for the newly selected rep.

When a user asks how some rep happened or was set, the data history view opens with that selected rep as the "Symptom" on the far right. If the selected rep is a data item, the process that created it and the data item inputs to that process are shown. If the selected rep is a process, its inputs are shown. Just like in the process history view, clicking the icon to the left of the rep button reveals more inputs and processes.

The user can close a view when it's not needed and it will be reopened when the corresponding question is asked. The views can also be reloaded, for instance when there's a problem in rendering a graph or if a button is not responding.

2.5.2 Woodstein's Views Include Buttons for Navigation

Each view include buttons in its title bar to help the user when interacting with the view. When scrolling around a document, it's easy to lose track of the selected rep. In all views, it can be brought into focus with the "Focus" button. Each view also has a "Back" button to reselect the previously selected rep.

The process history view includes buttons at the top for navigating through the process history.

Action Up with an arrow pointing up, selects the previous action or its last step if it has one

Action Down with an arrow down, select the next action, or step of this action if it has steps

User Up with "user" and an arrow up, goes to the previous user action

User Down with "user" and an arrow down, goes to the next user action

The data history view allows a user to reset the current symptom when the current graph becomes too large and unclear. Asking how another rep happened or was set reopens the data history view with the newly selected rep as the new symptom.

2.5.3 Browse During Browsing Mode, Inspect During Inspection Mode

During inspection mode with Woodstein, the user can only inspect page elements and not perform actions with them. A link can be inspected to see the context of the action of clicking it, but the user has to turn off inspection mode to actually follow the link.

2.6 Order Scenario: Placing an Order and Learning about Placing Orders with Woodstein

We've seen how Woodstein helps a user see the history and status of a process he's initiated. It is also helpful when a user is performing an action, particularly when the steps of the action are confusing. Woodstein shows the overall structure of the process so far, allowing the user to see what has already happened and what remains to be done.

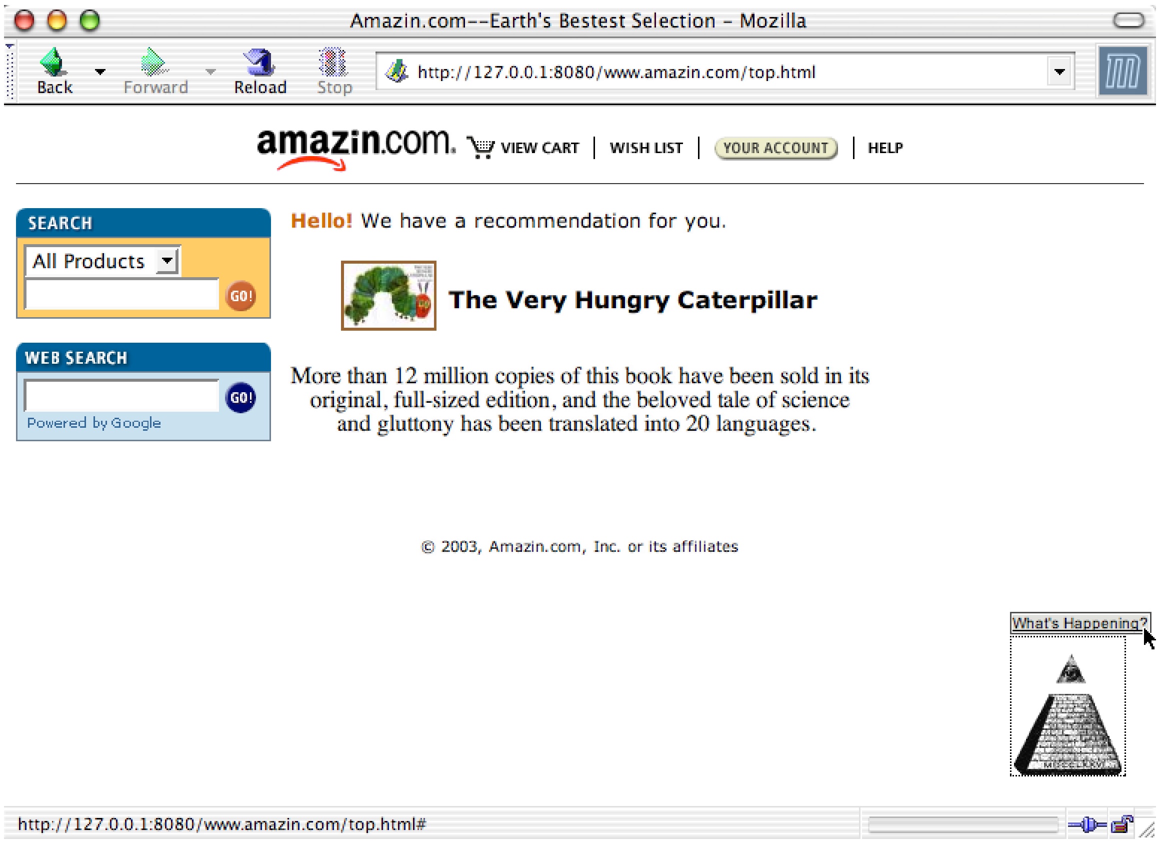
We will see how the user can use Woodstein to learn more about placing an order at an online retailer. In the previous example, we started after the user had already placed the order. Now we will go back to before the order was complete, while the user was placing it.

Our user has never placed an order at Amazin.com, but he knows he can get help from Woodstein. He begins by loading the Amazin.com front page and, by pressing down on the inspector icon, he selects its menu option to see "What's Happening?" (Figure 2-11).

Woodstein opens a new "what's happening" tracking view that, like the why view, shows the process history of the user's process. He can see that it recognized his first step, which has a black border since it was the last step he performed (Figure 2-12).

Now he turns back to browsing. He likes Amazin.com's book recommendation and thinks it would make a fine gift so he clicks on the image to go to the book's page. The new page loads (Figure 2-13) and Woodstein updates the tracking view (Figure 2-14).

The user is certain that he'd like to buy the book, but he's unsure about what to do next. He wants to get help with placing the order, and wants to know about the process of placing an order, so he turns to the tracking view and clicks on "You are placing an


Amazin.com--Earth's Bestest Selection - Mozilla
Back
Forward
Reload
Stop
http://127.0.0.1:8080/www.amazin.com/top.html
amazin.com
VIEW CART
WISH LIST
YOUR ACCOUNT
HELP
SEARCH
All Products
GO!
WEB SEARCH
GO!
Powered by Google
What's Happening?
http://127.0.0.1:8080/www.amazin.com/top.html#

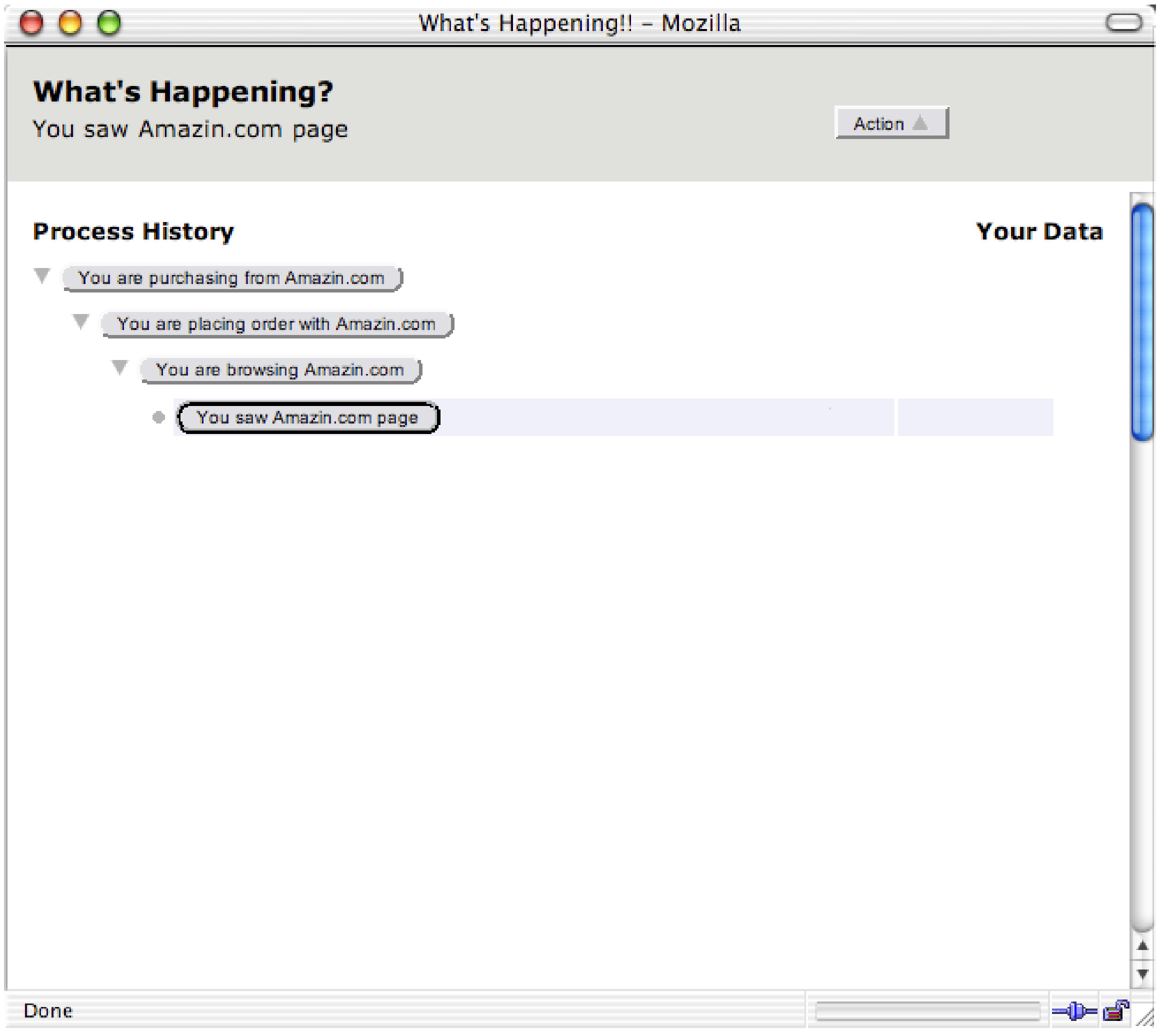
Hello! We have a recommendation for you.

The Very Hungry Caterpillar

More than 12 million copies of this book have been sold in its original, full-sized edition, and the beloved tale of science and gluttony has been translated into 20 languages.

© 2003, Amazin.com, Inc. or its affiliates

Figure 2-11: Asking "What's Happening?"


What's Happening!! – Mozilla

What's Happening?

You saw Amazin.com page

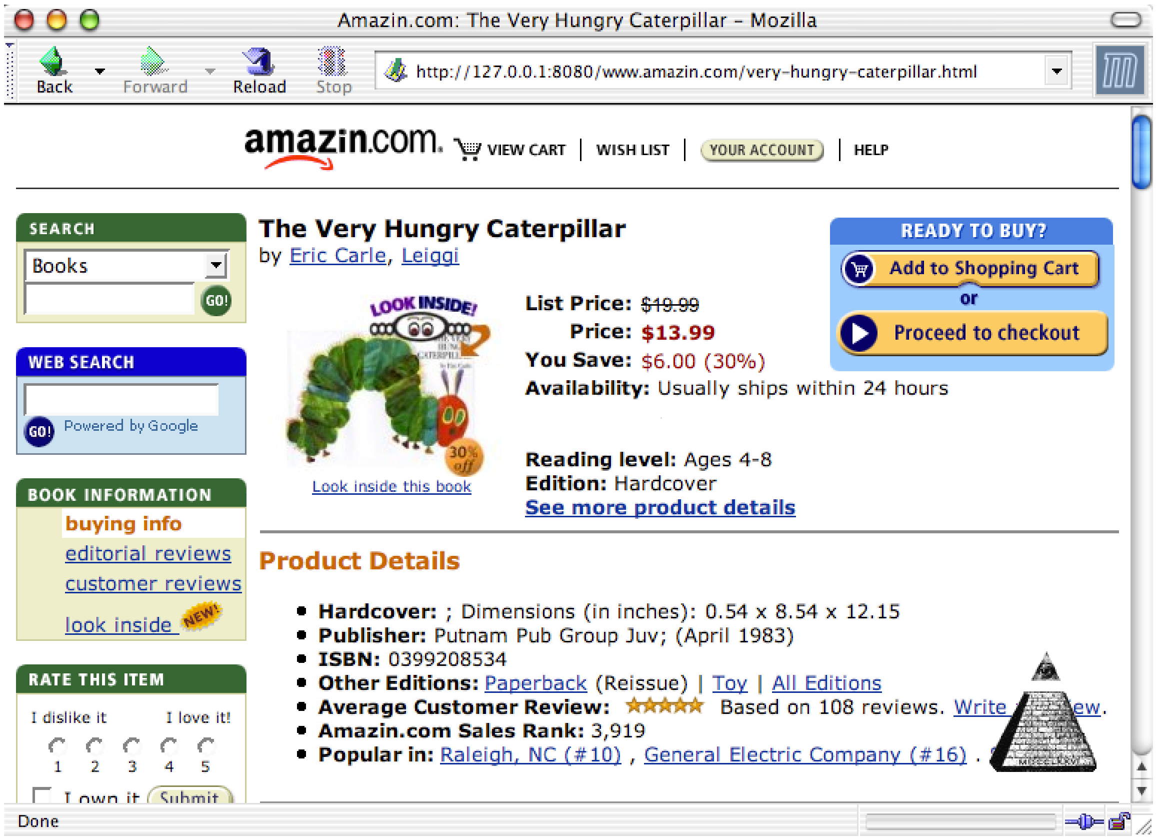
Process History

Your Data

- You are purchasing from Amazin.com
  - You are placing order with Amazin.com
    - You are browsing Amazin.com
      - You saw Amazin.com page

Done

Figure 2-12: Viewing Woodstein's tracking of a user action



Amazin.com: The Very Hungry Caterpillar - Mozilla

http://127.0.0.1:8080/www.amazin.com/very-hungry-caterpillar.html

The Very Hungry Caterpillar
by Eric Carle, Leiggi

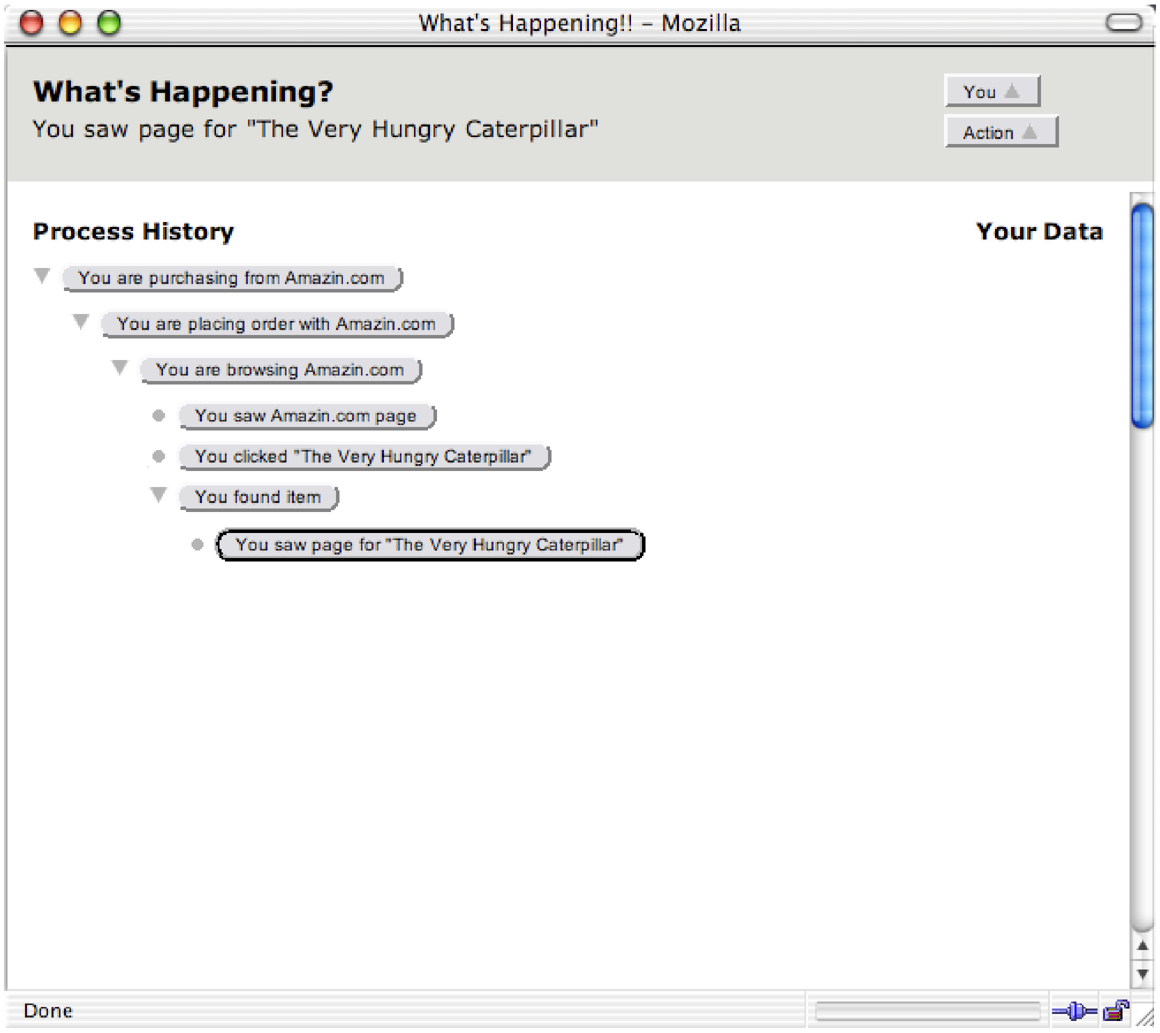
List Price: ~~$19.99~~
Price: $13.99
You Save: $6.00 (30%)
Availability: Usually ships within 24 hours

Reading level: Ages 4-8
Edition: Hardcover
See more product details

Product Details

- Hardcover: ; Dimensions (in inches): 0.54 x 8.54 x 12.15
- Publisher: Putnam Pub Group Juv; (April 1983)
- ISBN: 0399208534
- Other Editions: Paperback (Reissue) | Toy | All Editions
- Average Customer Review: ★★★★★ Based on 108 reviews. Write ew.
- Amazin.com Sales Rank: 3,919
- Popular in: Raleigh, NC (#10) , General Electric Company (#16) .

Figure 2-13: Browsing a book's item page


What's Happening!! – Mozilla

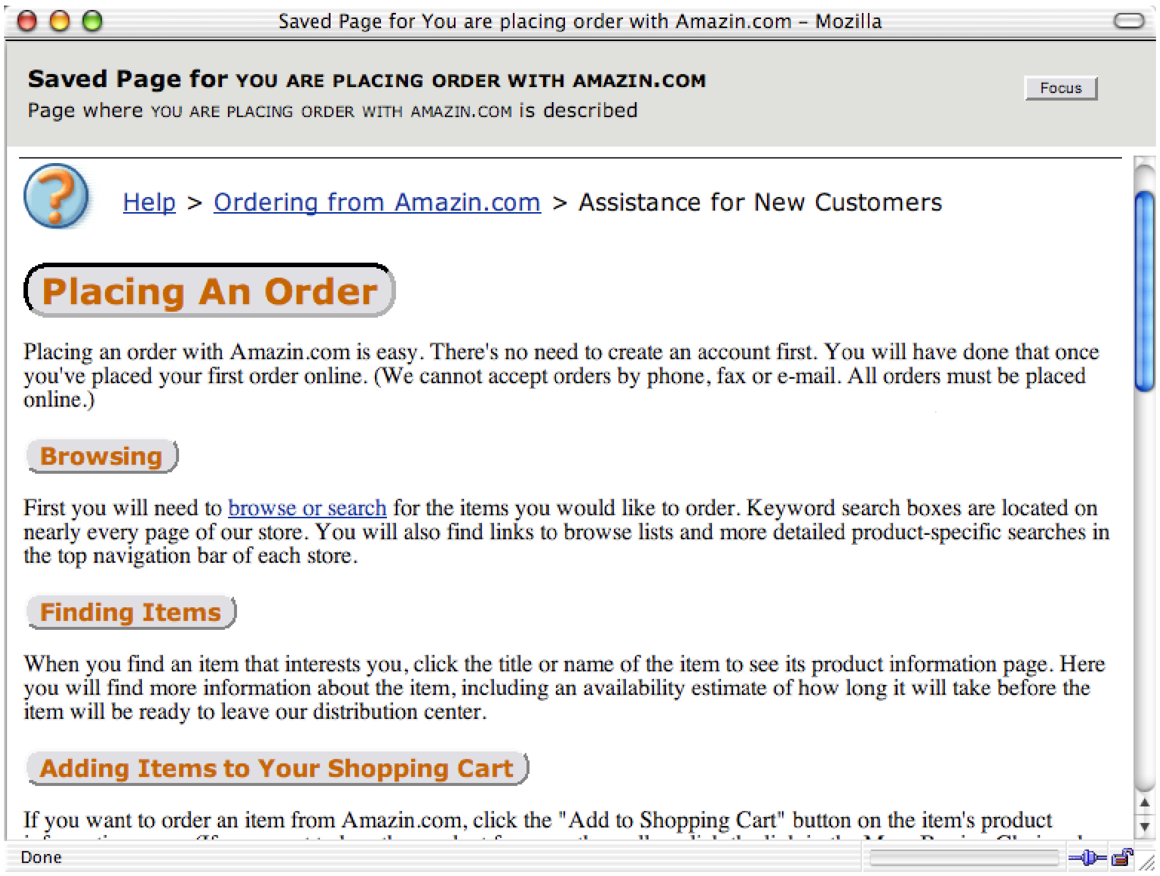
What's Happening?
You saw page for "The Very Hungry Caterpillar"

Process History

- You are purchasing from Amazin.com
  - You are placing order with Amazin.com
    - You are browsing Amazin.com
      - You saw Amazin.com page
      - You clicked "The Very Hungry Caterpillar"
      - You found item
        - You saw page for "The Very Hungry Caterpillar"

Your Data

Figure 2-14: Tracking the browsing of a book's item page



Saved Page for You are placing order with Amazin.com – Mozilla

Saved Page for YOU ARE PLACING ORDER WITH AMAZIN.COM

Page where YOU ARE PLACING ORDER WITH AMAZIN.COM is described

Help > Ordering from Amazin.com > Assistance for New Customers

Placing An Order

Placing an order with Amazin.com is easy. There's no need to create an account first. You will have done that once you've placed your first order online. (We cannot accept orders by phone, fax or e-mail. All orders must be placed online.)

Browsing

First you will need to browse or search for the items you would like to order. Keyword search boxes are located on nearly every page of our store. You will also find links to browse lists and more detailed product-specific searches in the top navigation bar of each store.

Finding Items

When you find an item that interests you, click the title or name of the item to see its product information page. Here you will find more information about the item, including an availability estimate of how long it will take before the item will be ready to leave our distribution center.

Adding Items to Your Shopping Cart

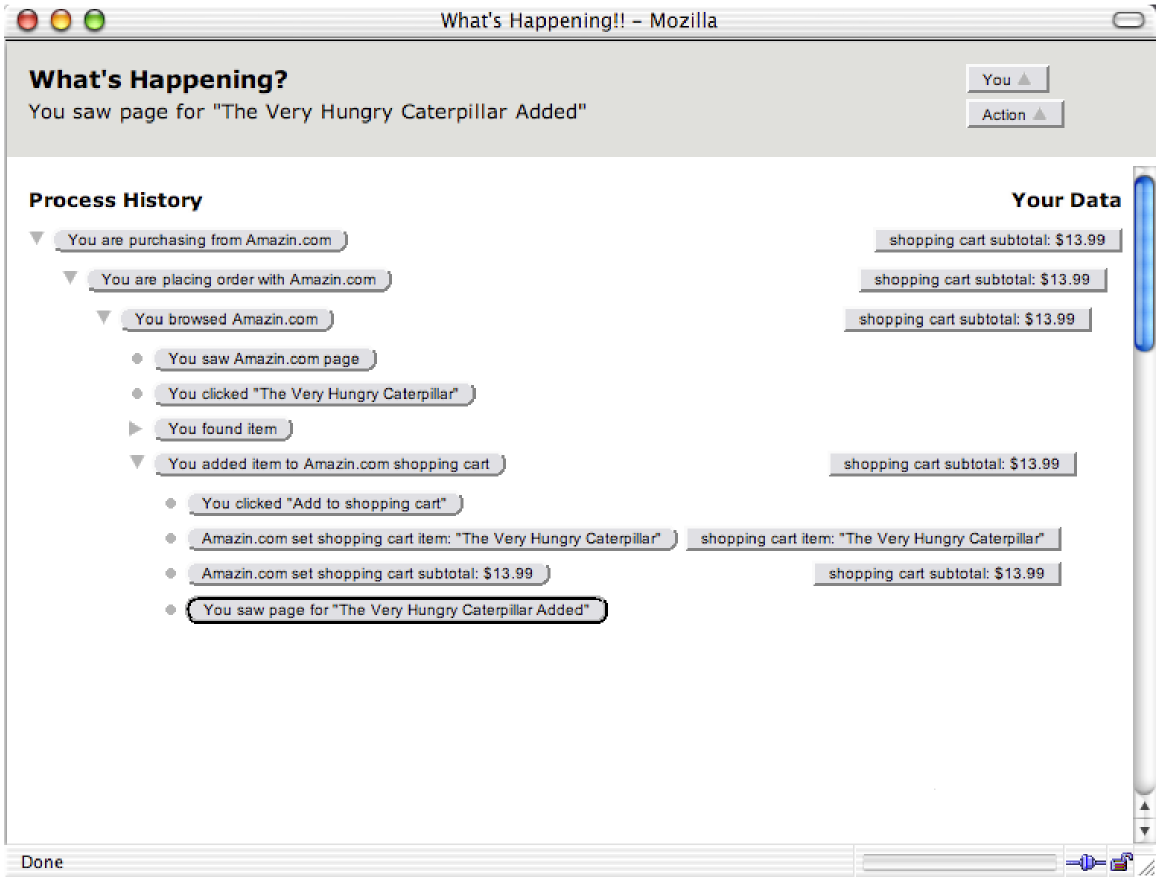
If you want to order an item from Amazin.com, click the "Add to Shopping Cart" button on the item's product

Figure 2-15: Viewing how to place an order

order with Amazin.com". That loads Woodstein's retrieved page describing the process of placing an order (Figure 2-15). He matches the steps with the steps he's gone through so far - browsing, then finding an item - and sees what the next step is - adding an item his shopping cart by clicking "Add to Shopping Cart".

He closes the saved page view, returns to the book's item page and clicks on the "Add to Shopping Cart" button. The page updates to show that the item is in his shopping cart, and the tracking view also updates to show that Woodstein saw both the user's action of adding the item, and Amazin.com's reaction of updating the shopping cart contents and subtotal (Figure 2-16).

Satisfied that he's now a pro at placing orders at Amazin.com, the user quickly finishes up his order so he can move on to his mid-afternoon break with some video-golf.


What's Happening!! – Mozilla

What's Happening?
You saw page for "The Very Hungry Caterpillar Added"

Process History

Your Data

- You are purchasing from Amazin.com — shopping cart subtotal: $13.99
  - You are placing order with Amazin.com — shopping cart subtotal: $13.99
    - You browsed Amazin.com — shopping cart subtotal: $13.99
      - You saw Amazin.com page
      - You clicked "The Very Hungry Caterpillar"
      - You found item
      - You added item to Amazin.com shopping cart — shopping cart subtotal: $13.99
        - You clicked "Add to shopping cart"
        - Amazin.com set shopping cart item: "The Very Hungry Caterpillar" — shopping cart item: "The Very Hungry Caterpillar"
        - Amazin.com set shopping cart subtotal: $13.99 — shopping cart subtotal: $13.99
        - You saw page for "The Very Hungry Caterpillar Added"

Done

Figure 2-16: Browsing a book's item page

2.6.1 How Woodstein Presents its Tracking of a User's Action

In this example, we've seen Woodstein's "what's happening" tracking view that, like its "why" process history view, shows the history of a process. Unlike the process history view, however, it shows the history of the user's current process. Recall that the process history view showed the history of the credit card transaction before. If the user had loaded the tracking view while checking his credit card transactions it would show him logging into his bank account and checking his credit card history.

As part of monitoring a user's action, Woodstein retrieves pages that describe the related processes. In this scenario, we saw how it loaded the help page at Amazin.com to describe the steps involved in placing an order. The user is able to get help with the action at the appropriate level by selecting a process to inspect.

2.7 How Woodstein Provides Help in Diagnosing Problems

We've seen how Woodstein can play an important role in helping collect information about a user's actions on the web. Its integrated view of actions and data spanning multiple web sites can help users instantly and easily diagnose problems. In addition to the task of collecting and compiling information, however, there are other dimensions to diagnosing and resolving problems on the web. The user must also sort through the information, and generate and test hypotheses for where the problem originated. Although this is a skill programmers develop by debugging, it's not something that end-users, who may not be accustomed to reasoning about formal processes, are necessarily familiar with. In this section, we will see how Woodstein guides the user in debugging by capturing his judgment of the correctness of data and processes that he sees.

2.7.1 Top-down and Bottom-up Debugging

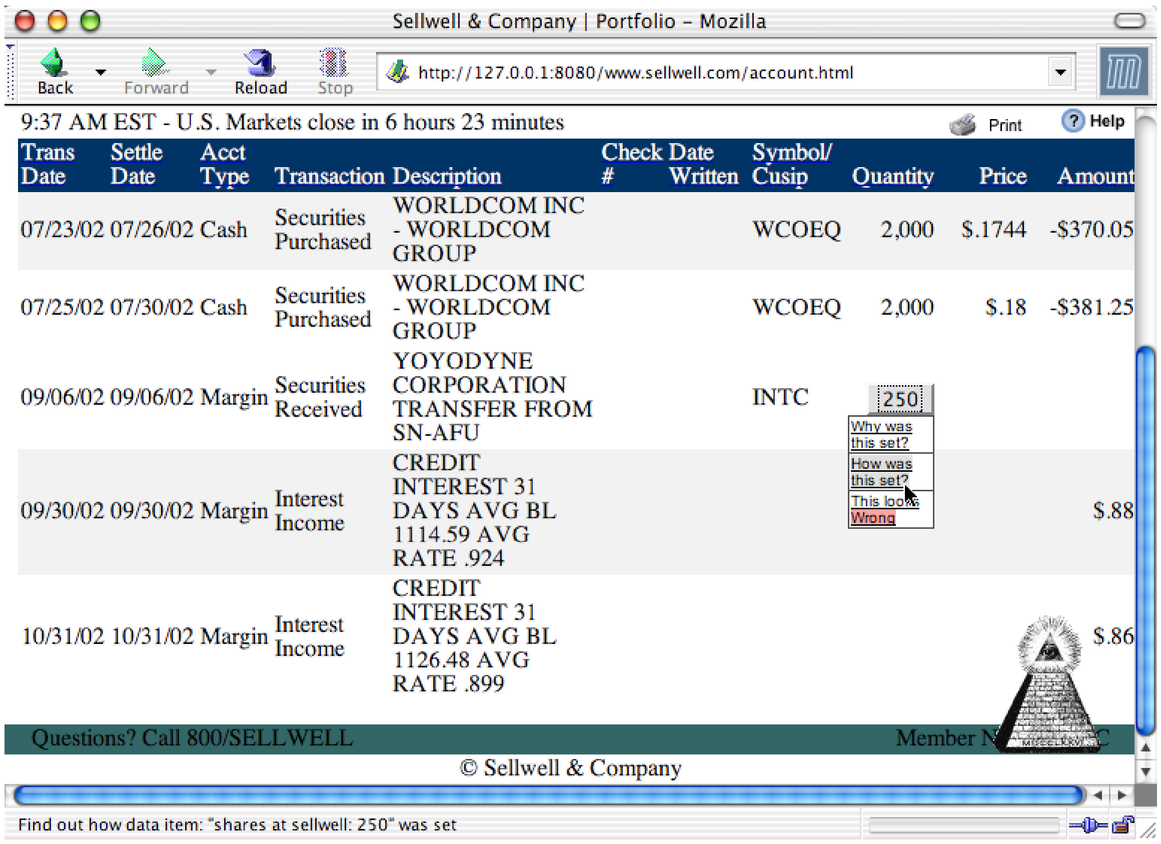
There are two different approaches that programmers take in debugging. They may debug in a "top-down" style by starting from a top-level process known to be faulty. Guided by the data they produce, the programmer isolates more and more specific processes until the buggy process or incorrect data is found. Another approach is "bottom-up" debugging in which the sources of incorrect data are iteratively tracked back until the process or data that caused the symptom is found.

Both of these approaches require that the programmer go through a “process of elimination”, eliminating possible causes that are found to be correct and examining ones that aren’t until the source of the problem is found. In the top-down approach, the data produced by the steps of the incorrect process are examined. When a process takes correct inputs but generates an incorrect output, it is inspected. In the bottom-up approach, the input values used to generate a value are examined. When all but one inputs are found to be correct, the remaining one must be incorrect and is inspected. Thus, even though they are employed differently, the top-down and bottom-up debugging styles both involve the process of elimination. The symptom is the root of a tree traversed one level at a time. In top-down debugging, the tree is the history of an overall process while in bottom-up debugging, the tree is the history of a data item. Woodstein’s process history and data history views allow top-down and bottom-up debugging styles, respectively.

2.7.2 Marking Incorrect Processes and Data

Even though it supports different debugging styles, the domains of programming and diagnosing problems on the web are different. End-users are not necessarily expert debuggers, and diagnosing a problem may extend over time as new actions and reactions occur. For these reasons, Woodstein manages a user’s intermediate discoveries while debugging and tracks his judgments about the correctness and incorrectness of data and processes.

Woodstein support a user in debugging by allowing him to mark data and process representations and guides him by mark further reps. A user can say that some data item “looks wrong” or a process “looks unsuccessful”, and Woodstein creates an annotation of “incorrect” for the rep, rendering it in red. It then looks at the data and processes that caused the rep in both the data history and process history, and marks them as “maybe incorrect”, rendering them in yellow. The user identifies the correct causes by noting that each “looks correct”, causing Woodstein to render them in green. As the user continues to judge processes and data as correct or incorrect, Woodstein guides him in isolating the problem to a single process or data rep.


Sellwell & Company | Portfolio – Mozilla

http://127.0.0.1:8080/www.sellwell.com/account.html

9:37 AM EST - U.S. Markets close in 6 hours 23 minutes

Trans Date	Settle Date	Acct Type	Transaction	Description	Check #	Date Written	Symbol/ Cusip	Quantity	Price	Amount
07/23/02	07/26/02	Cash	Securities Purchased	WORLDCOM INC - WORLDCOM GROUP			WCOEQ	2,000	$.1744	-$370.05
07/25/02	07/30/02	Cash	Securities Purchased	WORLDCOM INC - WORLDCOM GROUP			WCOEQ	2,000	$.18	-$381.25
09/06/02	09/06/02	Margin	Securities Received	YOYODYNE CORPORATION TRANSFER FROM SN-AFU			INTC	250		
09/30/02	09/30/02	Margin	Interest Income	CREDIT INTEREST 31 DAYS AVG BL 1114.59 AVG RATE .924						$.88
10/31/02	10/31/02	Margin	Interest Income	CREDIT INTEREST 31 DAYS AVG BL 1126.48 AVG RATE .899						$.86

Questions? Call 800/SELLWELL

Member N

© Sellwell & Company

Find out how data item: "shares at sellwell: 250" was set

Figure 2-17: Asking how a stock transfer happened

2.8 Stock Scenario: Inspecting a Stock Purchase With Woodstein

We can see how Woodstein guides a user through the process of diagnosing a problem with another example. In this scenario, the user is an employee at Yoyodyne and is enrolled in its share purchase plan. Every pay period, it sets aside a portion of his salary money in an account. Once a year it buys stock for him at a discounted price from its broker, SN-AFU. He has set up his account at SN-AFU to automatically transfer the stock to his broker, Sellwell. Now he's looking at the most recent transfer at Sellwell and it looks lower than the usual amount purchased (Figure 2-17).

He wants to know how it was set, so he loads the data history view. He'd like help from Woodstein in diagnosing the problem, so he notes that the stock at his broker looks wrong (Figure 2-19).

Woodstein updates the data history view, marking the preceding process, "Sellwell set

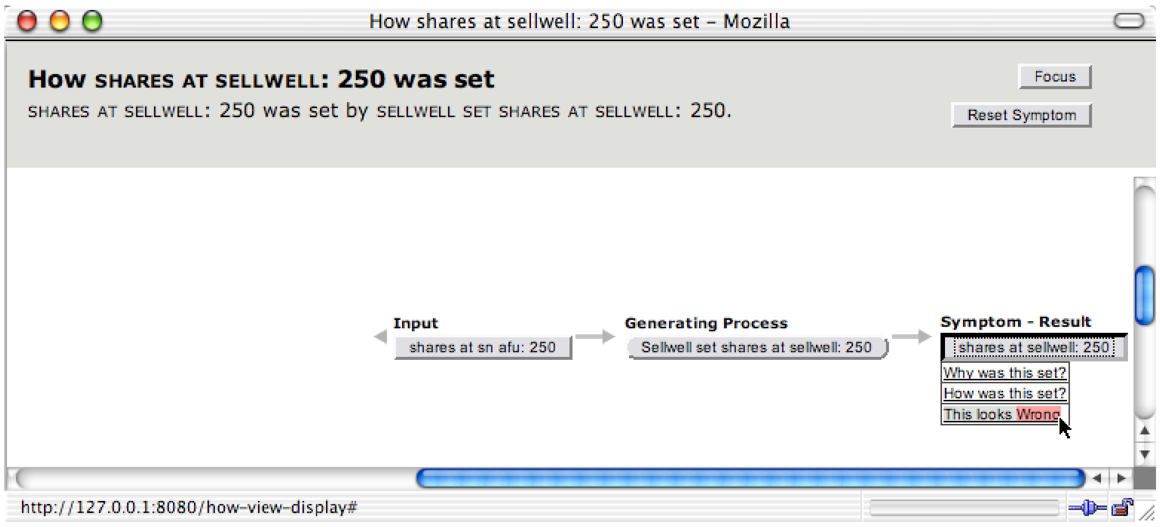

How shares at sellwell: 250 was set – Mozilla
How SHARES AT SELLWELL: 250 was set
SHARES AT SELLWELL: 250 was set by SELLWELL SET SHARES AT SELLWELL: 250.
Focus
Reset Symptom
Input
shares at sn afu: 250
Generating Process
Sellwell set shares at sellwell: 250
Symptom - Result
shares at sellwell: 250
Why was this set?
How was this set?
This looks Wrong
http://127.0.0.1:8080/how-view-display#

Figure 2-18: Noting that the amount of stock transferred looks wrong

shares at sellwell: 250" to yellow for "maybe wrong". It also marks the input for that process, "shares at sn-afu: 250" to "maybe wrong" (Figure 2-19).

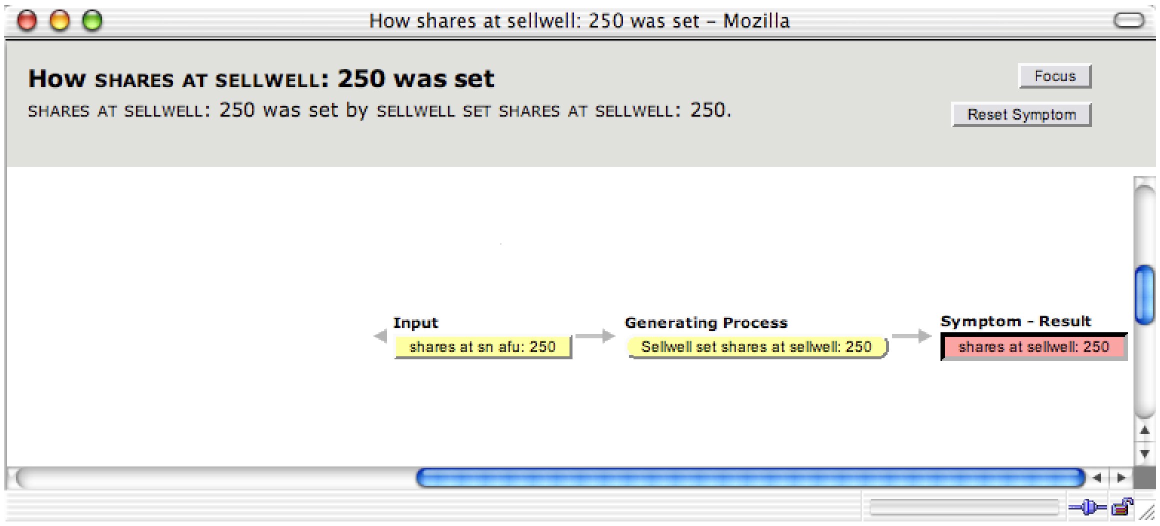
Woodstein also opens a new view, the "debugging trail" view to help the user keep track of his process in diagnosing the problem. Moving the mouse over the rep "Sellwell set shares at sellwell: 250", under "Next Step" tells him how to determine whether the rep is correct (Figure 2-20).

It looks like the correct number of shares were transferred from SN-AFU, so the user goes back to the data history view and notes that "Sellwell set shares at sellwell: 250" looks correct (Figure 2-21).

Woodstein automatically updates the data history view, marking the data item input and process that computed "shares at sn-afu: 250" as maybe incorrect. The user scrolls to see them better (Figure 2-22).

The user continues, finding that the setting of the shares at SN-AFU was successful and eventually tracing the problem back to Yoyodyne's original calculation of the number of shares to purchase. He notes that the share purchase budget of $5000 divided by the share price of $20 is 250, so it looks like the correct number of shares were purchased given the budget and price. He marks the setting of the shares to purchase as looking correct (Figure 2-23).

He can see in the page (Figure 2-24) that the purchase price is $20, and it's %80 of the lower of the purchase data value and the grant date value. %80 of $25 is $20, so that looks

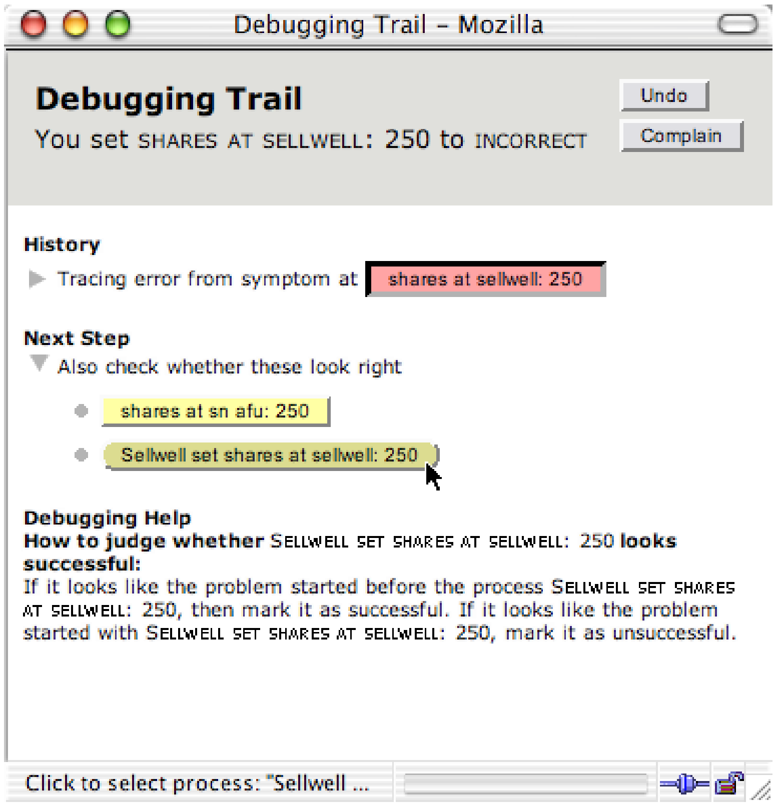


How shares at sellwell: 250 was set

shares at sellwell: 250 was set by sellwell set shares at sellwell: 250.

shares at sn afu: 250 → Sellwell set shares at sellwell: 250 → shares at sellwell: 250

Figure 2-19: Viewing how the amount of stock transferred was set after noting that it looks wrong



Debugging Trail

You set shares at sellwell: 250 to incorrect

History

▶ Tracing error from symptom at shares at sellwell: 250

Next Step

▼ Also check whether these look right

- shares at sn afu: 250
- Sellwell set shares at sellwell: 250

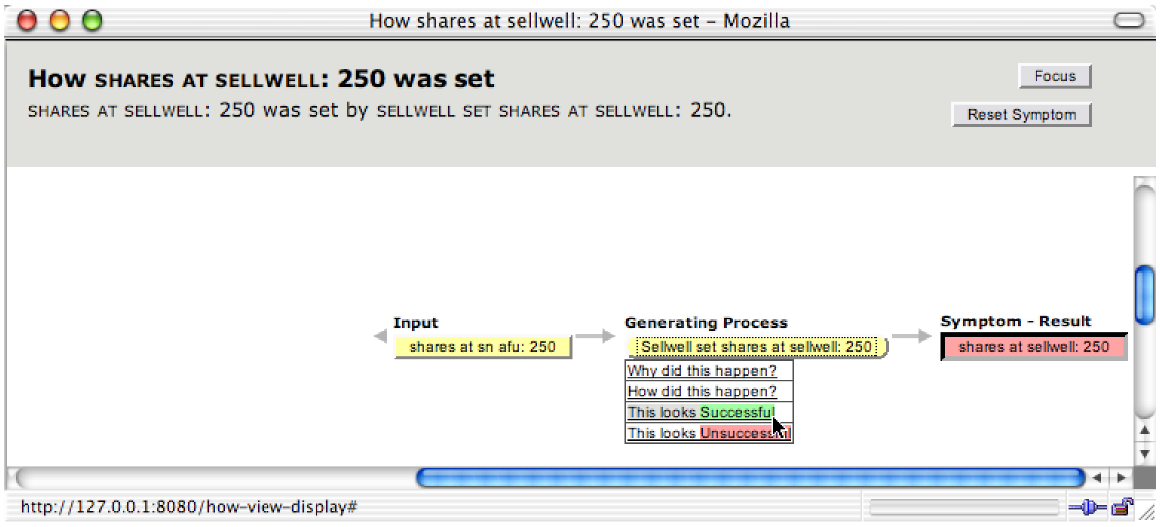
Debugging Help

How to judge whether Sellwell set shares at sellwell: 250 **looks successful:**

If it looks like the problem started before the process Sellwell set shares at sellwell: 250, then mark it as successful. If it looks like the problem started with Sellwell set shares at sellwell: 250, mark it as unsuccessful.

Click to select process: "Sellwell ...

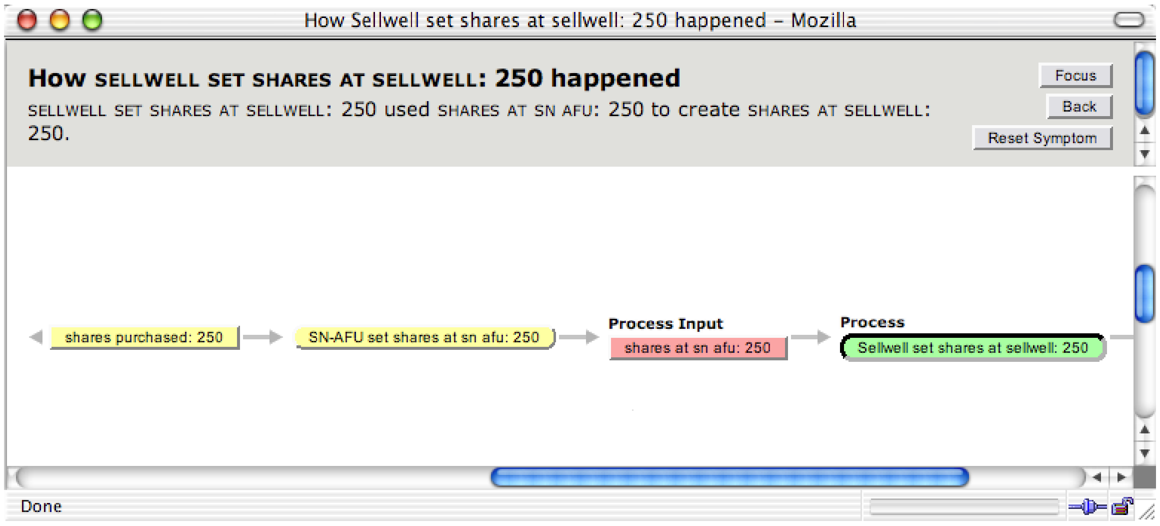
Figure 2-20: Viewing the debugging trail


How shares at sellwell: 250 was set – Mozilla

How SHARES AT SELLWELL: 250 was set

SHARES AT SELLWELL: 250 was set by SELLWELL SET SHARES AT SELLWELL: 250.

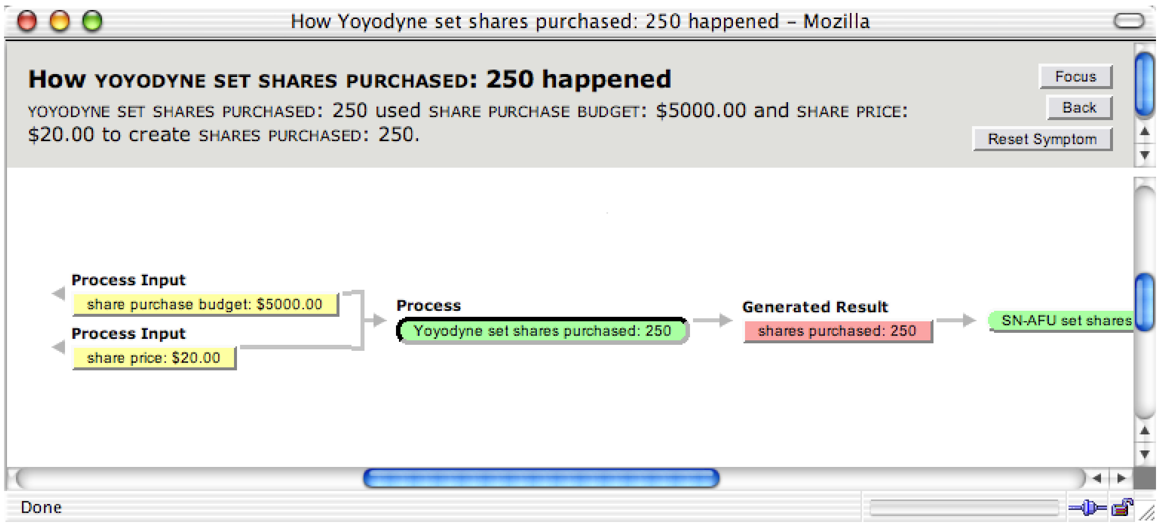
Figure 2-21: Noting that the setting of the number of shares at Sellwell looks successful


How Sellwell set shares at sellwell: 250 happened – Mozilla

How SELLWELL SET SHARES AT SELLWELL: 250 happened

SELLWELL SET SHARES AT SELLWELL: 250 used SHARES AT SN AFU: 250 to create SHARES AT SELLWELL: 250.

Figure 2-22: Viewing how Sellwell set the shares at Sellwell the correctness of its input


How Yoyodyne set shares purchased: 250 happened – Mozilla

How YOYODYNE SET SHARES PURCHASED: 250 happened

YOYODYNE SET SHARES PURCHASED: 250 used SHARE PURCHASE BUDGET: $5000.00 and SHARE PRICE: $20.00 to create SHARES PURCHASED: 250.

Figure 2-23: Noting that Yoyodyne's calculation of the number to purchase looks correct

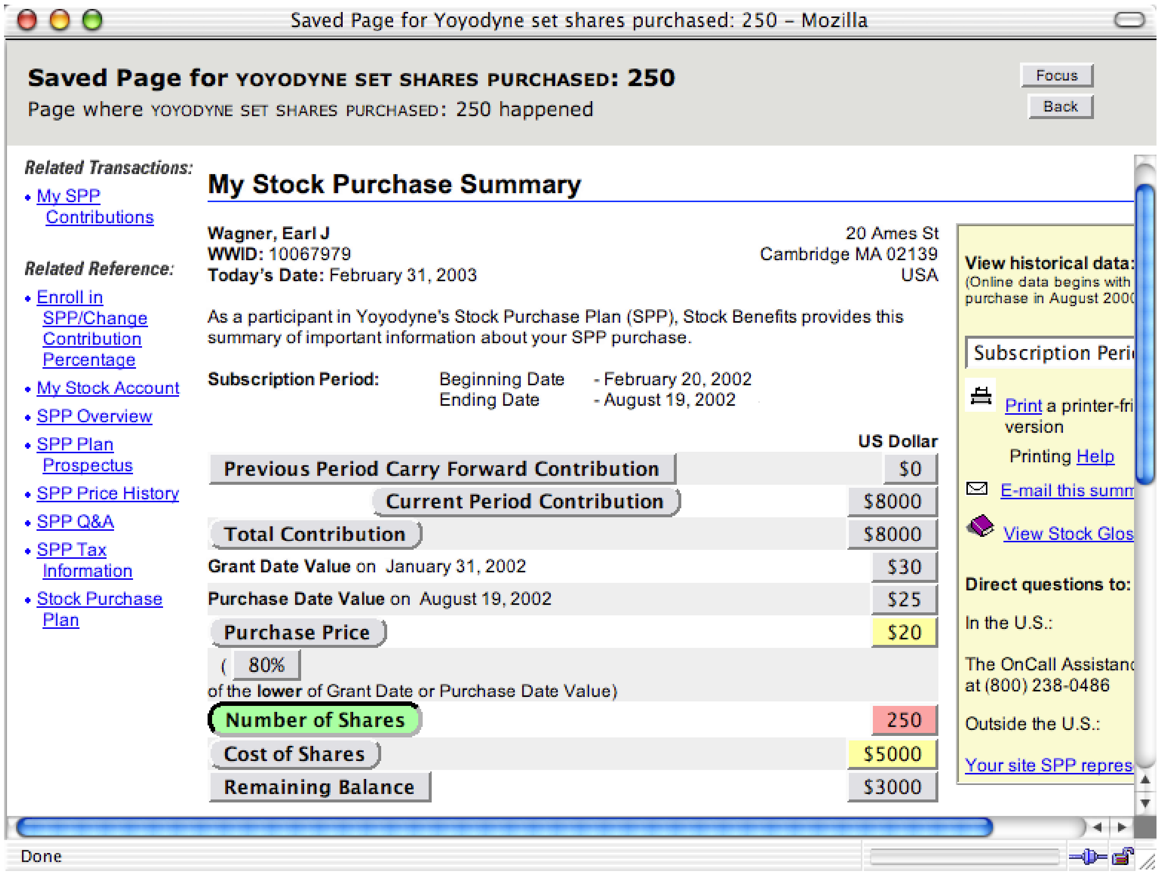
right and he marks the share price correct. Woodstein isolates the problem to the share purchase budget. The user sees that the preceding process took both his total contribution and the IRS limit as inputs. The total contribution looks like what he would expect to see spent, so now he wants to investigate the IRS limit and he clicks on it (Figure 2-25).

Selecting the IRS limit loads the relevant saved page in the saved page view (Figure 2-26). Within the page, he sees why the amount of stock purchased was so low - Yoyodyne imposed the IRS limit of $5000. The rest will be used in the future.

2.8.1 Summary

In this example, the user took advantage of Woodstein's support for bottom-up debugging with the data history view in identifying the process that behaved unexpectedly - the application of the IRS limit.

In this example, the user thought the web site caused an error, but in fact the unexpected result was due to an obscure policy that the user wasn't aware of. Woodstein still helped the user, however, by enabling him to compare his mental model of what should happen with a record of what actually did happen, then identify where his mental model was incorrect. It not only showed the abstract description of the obscure policy, but also gave a concrete example of how it worked by showing how it affected the user's data making the learning experience even more personalized.



Saved Page for Yoyodyne set shares purchased: 250 – Mozilla

Saved Page for YOYODYNE SET SHARES PURCHASED: 250
Page where YOYODYNE SET SHARES PURCHASED: 250 happened

Focus
Back

Related Transactions:
- My SPP Contributions

Related Reference:
- Enroll in SPP/Change Contribution Percentage
- My Stock Account
- SPP Overview
- SPP Plan Prospectus
- SPP Price History
- SPP Q&A
- SPP Tax Information
- Stock Purchase Plan

My Stock Purchase Summary

Wagner, Earl J
WWID: 10067979
Today's Date: February 31, 2003

20 Ames St
Cambridge MA 02139
USA

As a participant in Yoyodyne's Stock Purchase Plan (SPP), Stock Benefits provides this summary of important information about your SPP purchase.

Subscription Period: Beginning Date - February 20, 2002
Ending Date - August 19, 2002

	US Dollar
Previous Period Carry Forward Contribution	$0
Current Period Contribution	$8000
Total Contribution	$8000
Grant Date Value on January 31, 2002	$30
Purchase Date Value on August 19, 2002	$25
Purchase Price	$20
(80% of the **lower** of Grant Date or Purchase Date Value)	
Number of Shares	250
Cost of Shares	$5000
Remaining Balance	$3000

View historical data:
(Online data begins with purchase in August 200

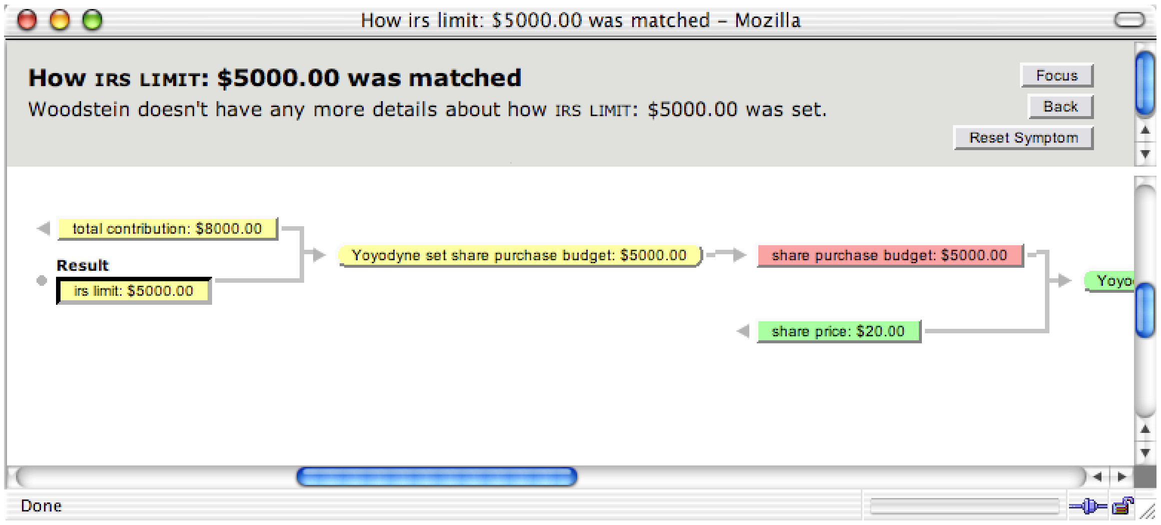
Subscription Peri

Print a printer-fri version
Printing Help
E-mail this summ
View Stock Glos

Direct questions to:
In the U.S.:
The OnCall Assistanc at (800) 238-0486
Outside the U.S.:
Your site SPP repres

Done

Figure 2-24: Saved Page view for "Yoyodyne set shares purchased: 250"



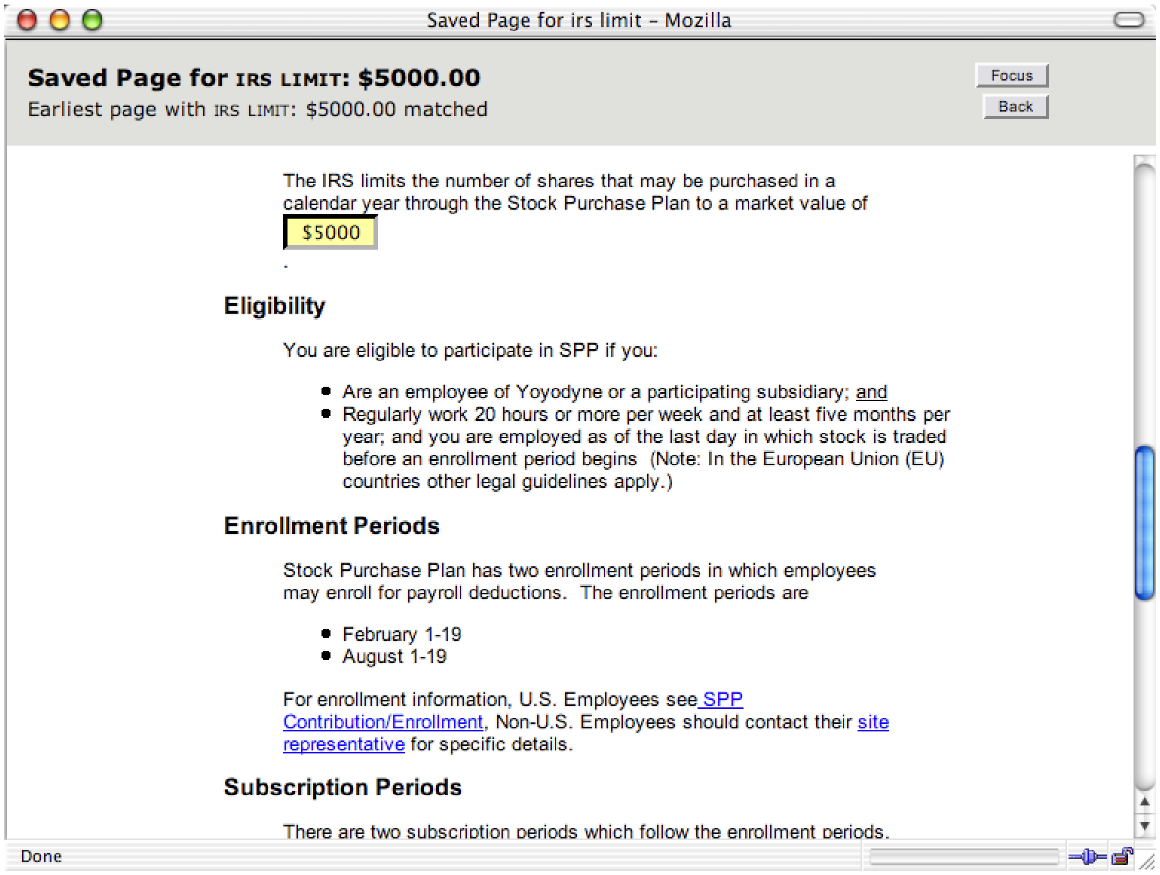
How irs limit: $5000.00 was matched – Mozilla

How IRS LIMIT: $5000.00 was matched
Woodstein doesn't have any more details about how IRS LIMIT: $5000.00 was set.

Focus
Back
Reset Symptom

total contribution: $8000.00
Result
irs limit: $5000.00
Yoyodyne set share purchase budget: $5000.00
share purchase budget: $5000.00
share price: $20.00
Yoyo
Done

Figure 2-25: Data history view with "IRS limit" selected



Saved Page for IRS LIMIT: $5000.00
Earliest page with IRS LIMIT: $5000.00 matched

The IRS limits the number of shares that may be purchased in a calendar year through the Stock Purchase Plan to a market value of $5000.

Eligibility

You are eligible to participate in SPP if you:

- Are an employee of Yoyodyne or a participating subsidiary; and
- Regularly work 20 hours or more per week and at least five months per year; and you are employed as of the last day in which stock is traded before an enrollment period begins (Note: In the European Union (EU) countries other legal guidelines apply.)

Enrollment Periods

Stock Purchase Plan has two enrollment periods in which employees may enroll for payroll deductions. The enrollment periods are

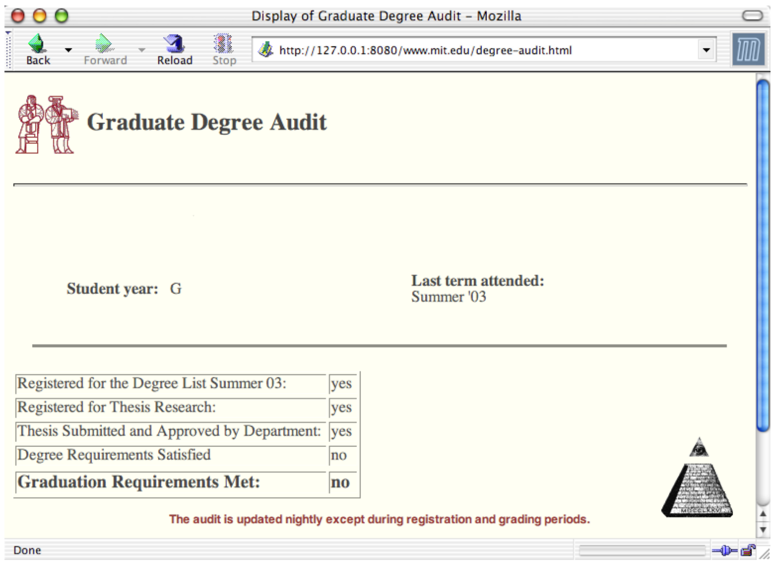
- February 1-19
- August 1-19

For enrollment information, U.S. Employees see SPP Contribution/Enrollment, Non-U.S. Employees should contact their site representative for specific details.

Subscription Periods

There are two subscription periods which follow the enrollment periods.

Figure 2-26: Data history view with “IRS limit” selected


Display of Graduate Degree Audit – Mozilla

http://127.0.0.1:8080/www.mit.edu/degree-audit.html

Graduate Degree Audit

Student year: G

Last term attended:
Summer '03

Registered for the Degree List Summer 03:	yes
Registered for Thesis Research:	yes
Thesis Submitted and Approved by Department:	yes
Degree Requirements Satisfied	no
Graduation Requirements Met:	**no**

The audit is updated nightly except during registration and grading periods.

Done

Figure 2-27: Viewing the graduation degree audit page

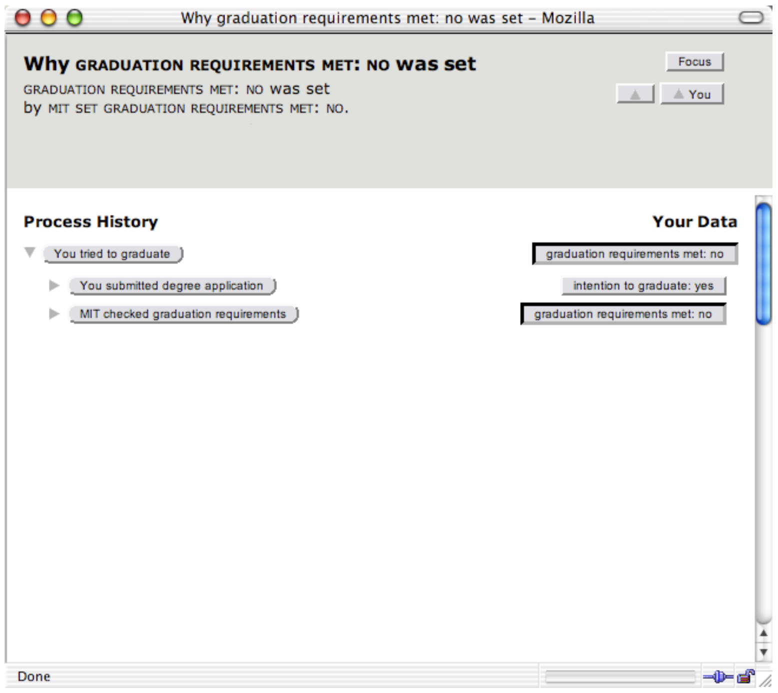
2.9 Graduation Scenario: Inspecting a Student's Graduation Status With Woodstein

We have seen how Woodstein visualizes processes that result from interactions between by a user and web site. It is also able to explain the processing a web site performs, which is particularly helpful for understanding the conclusions that result from the application of more complicated policies. With a model of the web site's processes, it collects information from pages, simulates the intermediate computations the web site performs, and explains how the result was generated. We can see how this works with an example in which a student intends to graduate. We will also see how a problem can be diagnosed in a top-down approach.

In this example, a masters' student is trying to graduate from the Institute he attends. He was sure to add himself to the list of graduating students earlier in the semester and he knows he's completed all other requirements for graduation. He knows that sometimes there are bureaucratic mistakes, however, so he goes online and checks the student information web site to verify that everything is OK.

Upon loading the graduated degree audit page, however, he sees that everything is not OK; it appears he's not eligible to graduate (Figure 2-27).

Fortunately, he's running Woodstein in the background and he turns on the inspector to see what's going on. He selects the "no" for "graduation requirements met" and asks


Why graduation requirements met: no was set – Mozilla

Why GRADUATION REQUIREMENTS MET: NO was set

GRADUATION REQUIREMENTS MET: NO was set
by MIT SET GRADUATION REQUIREMENTS MET: NO.

Process History	Your Data
You tried to graduate	graduation requirements met: no
You submitted degree application	intention to graduate: yes
MIT checked graduation requirements	graduation requirements met: no

Figure 2-28: Asking why graduation requirements were not met

“Why was this set?” and sees the process history view window in Figure 2-28.

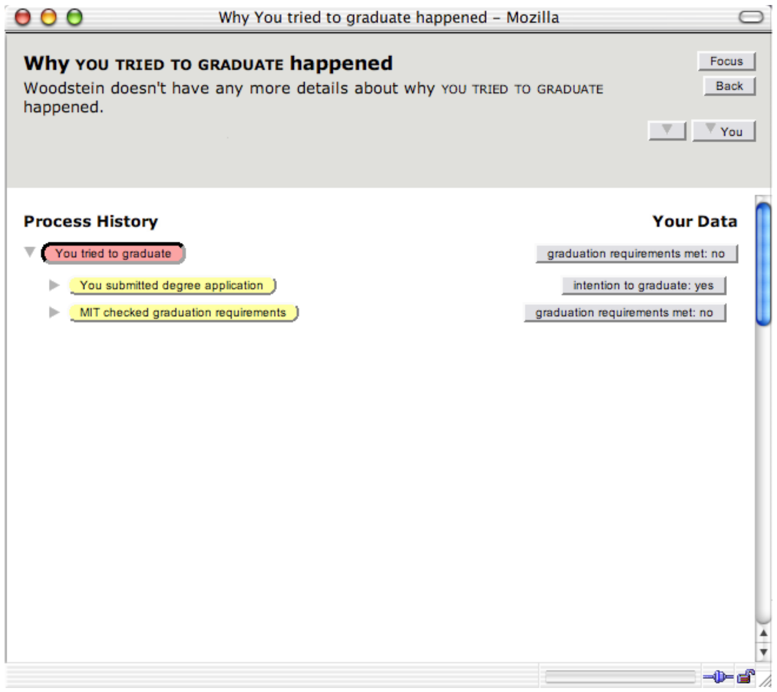
He sees “You tried to graduate” and marks it as looking unsuccessful. Woodstein opens the debugging trail view and updates the “why” process history view to indicate its steps that may also be unsuccessful (Figure 2-29).

He can see that his Institute received his degree application since it has “intention to graduate: yes”, so he marks that as successful. Woodstein automatically identifies the graduation requirements check as unsuccessful and sets its steps as possibly unsuccessful as well (Figure 2-30).

The student sees that all degree holds have been successfully cleared, but sees that his degree requirements were not satisfied and since this happened before his graduation requirements were not met, he marks the graduation requirements check as successful and Woodstein automatically marks the degree requirements check unsuccessful (Figure 2-31).

He sees that both his subject requirements and residency requirement were satisfied, and noting that the thesis requirement is not satisified, he sets the degree requirement check as successful. Woodstein marks the thesis requirements check as incorrect and its steps as maybe incorrect (Figure 2-32).

He sees that he successfully registered for research, that his thesis has been completed

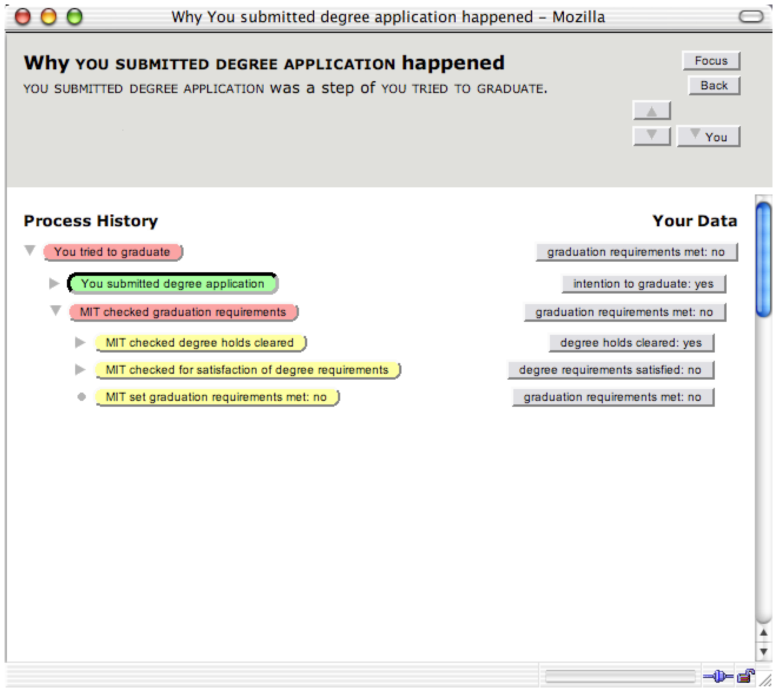

Why You tried to graduate happened – Mozilla

Why YOU TRIED TO GRADUATE happened

Woodstein doesn't have any more details about why YOU TRIED TO GRADUATE happened.

Process History	Your Data
You tried to graduate	graduation requirements met: no
You submitted degree application	intention to graduate: yes
MIT checked graduation requirements	graduation requirements met: no

Figure 2-29: Viewing why trying to graduate was unsuccessful and the possibly unsuccessful steps

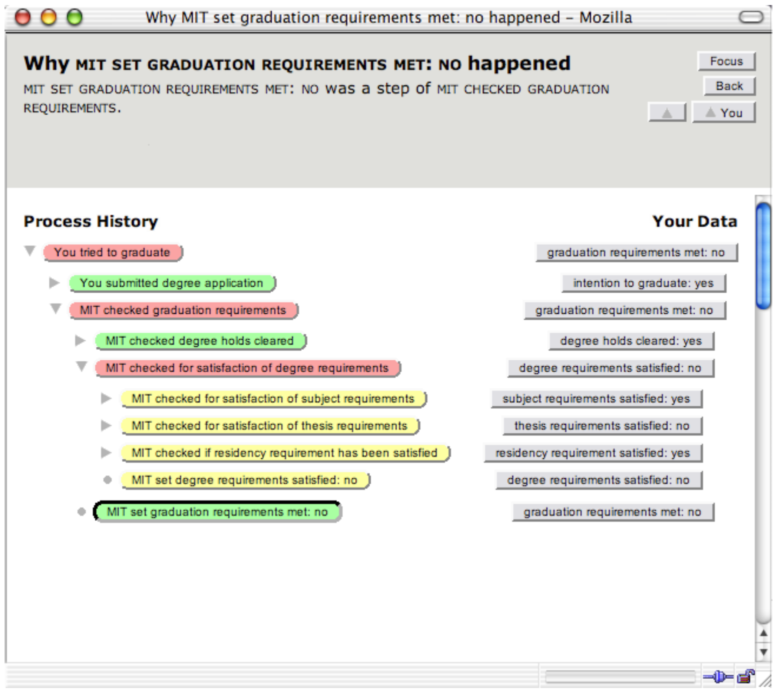

Why You submitted degree application happened – Mozilla

Why YOU SUBMITTED DEGREE APPLICATION happened

YOU SUBMITTED DEGREE APPLICATION was a step of YOU TRIED TO GRADUATE.

Process History	Your Data
You tried to graduate	graduation requirements met: no
You submitted degree application	intention to graduate: yes
MIT checked graduation requirements	graduation requirements met: no
MIT checked degree holds cleared	degree holds cleared: yes
MIT checked for satisfaction of degree requirements	degree requirements satisfied: no
MIT set graduation requirements met: no	graduation requirements met: no

Figure 2-30: Viewing why the graduation requirements were unsuccessfully checked and the possibly unsuccessful steps

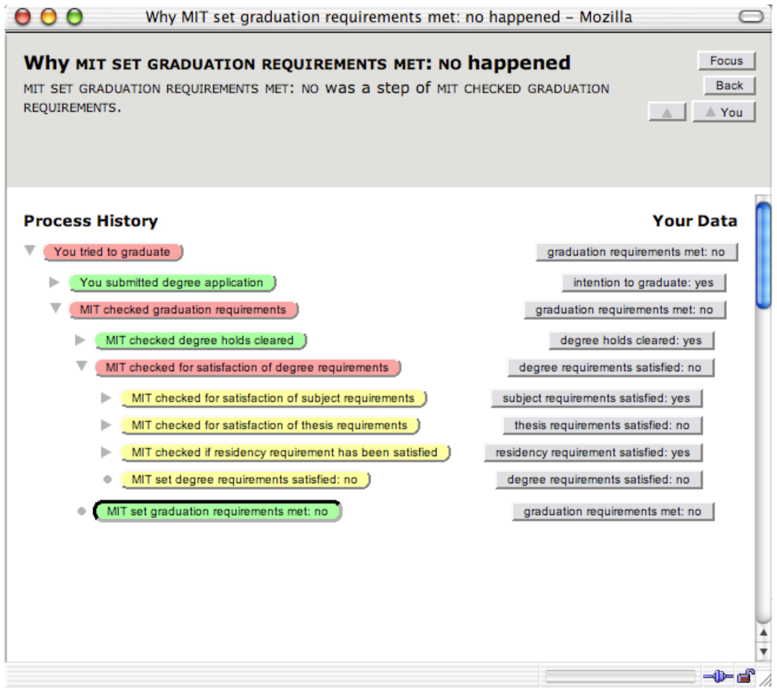

Why MIT set graduation requirements met: no happened – Mozilla

Why MIT SET GRADUATION REQUIREMENTS MET: NO happened

MIT SET GRADUATION REQUIREMENTS MET: NO was a step of MIT CHECKED GRADUATION REQUIREMENTS.

Process History	Your Data
You tried to graduate	graduation requirements met: no
You submitted degree application	intention to graduate: yes
MIT checked graduation requirements	graduation requirements met: no
MIT checked degree holds cleared	degree holds cleared: yes
MIT checked for satisfaction of degree requirements	degree requirements satisfied: no
MIT checked for satisfaction of subject requirements	subject requirements satisfied: yes
MIT checked for satisfaction of thesis requirements	thesis requirements satisfied: no
MIT checked if residency requirement has been satisfied	residency requirement satisfied: yes
MIT set degree requirements satisfied: no	degree requirements satisfied: no
MIT set graduation requirements met: no	graduation requirements met: no

Figure 2-31: Viewing why the degree requirements were unsuccessfully checked and the possibly unsuccessful steps

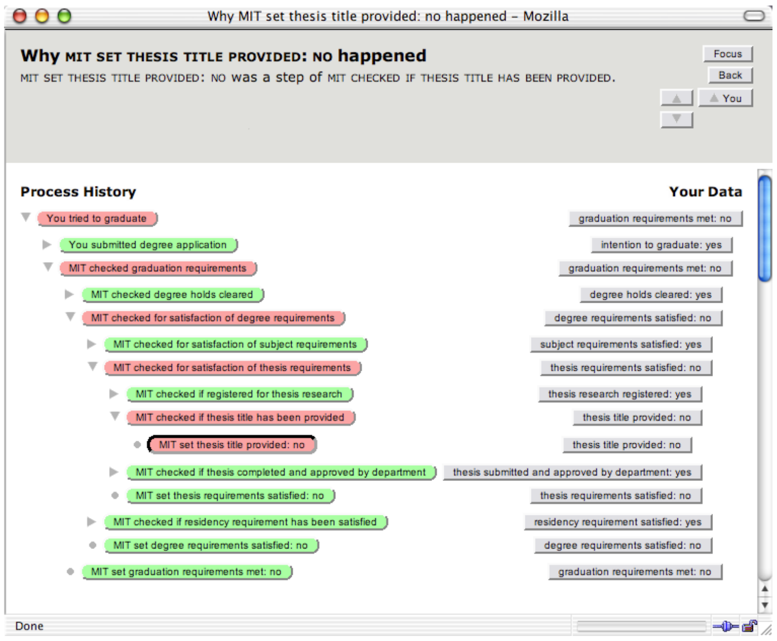

Why MIT set graduation requirements met: no happened – Mozilla

Why MIT SET GRADUATION REQUIREMENTS MET: NO happened

MIT SET GRADUATION REQUIREMENTS MET: NO was a step of MIT CHECKED GRADUATION REQUIREMENTS.

Process History	Your Data
You tried to graduate	graduation requirements met: no
You submitted degree application	intention to graduate: yes
MIT checked graduation requirements	graduation requirements met: no
MIT checked degree holds cleared	degree holds cleared: yes
MIT checked for satisfaction of degree requirements	degree requirements satisfied: no
MIT checked for satisfaction of subject requirements	subject requirements satisfied: yes
MIT checked for satisfaction of thesis requirements	thesis requirements satisfied: no
MIT checked if residency requirement has been satisfied	residency requirement satisfied: yes
MIT set degree requirements satisfied: no	degree requirements satisfied: no
MIT set graduation requirements met: no	graduation requirements met: no

Figure 2-32: Viewing why the thesis requirements were unsuccessfully checked and the possibly unsuccessful steps



Why MIT SET THESIS TITLE PROVIDED: NO happened

MIT SET THESIS TITLE PROVIDED: NO was a step of MIT CHECKED IF THESIS TITLE HAS BEEN PROVIDED.

Process History	Your Data
You tried to graduate	graduation requirements met: no
You submitted degree application	intention to graduate: yes
MIT checked graduation requirements	graduation requirements met: no
MIT checked degree holds cleared	degree holds cleared: yes
MIT checked for satisfaction of degree requirements	degree requirements satisfied: no
MIT checked for satisfaction of subject requirements	subject requirements satisfied: yes
MIT checked for satisfaction of thesis requirements	thesis requirements satisfied: no
MIT checked if registered for thesis research	thesis research registered: yes
MIT checked if thesis title has been provided	thesis title provided: no
MIT set thesis title provided: no	thesis title provided: no
MIT checked if thesis completed and approved by department	thesis submitted and approved by department: yes
MIT set thesis requirements satisfied: no	thesis requirements satisfied: no
MIT checked if residency requirement has been satisfied	residency requirement satisfied: yes
MIT set degree requirements satisfied: no	degree requirements satisfied: no
MIT set graduation requirements met: no	graduation requirements met: no

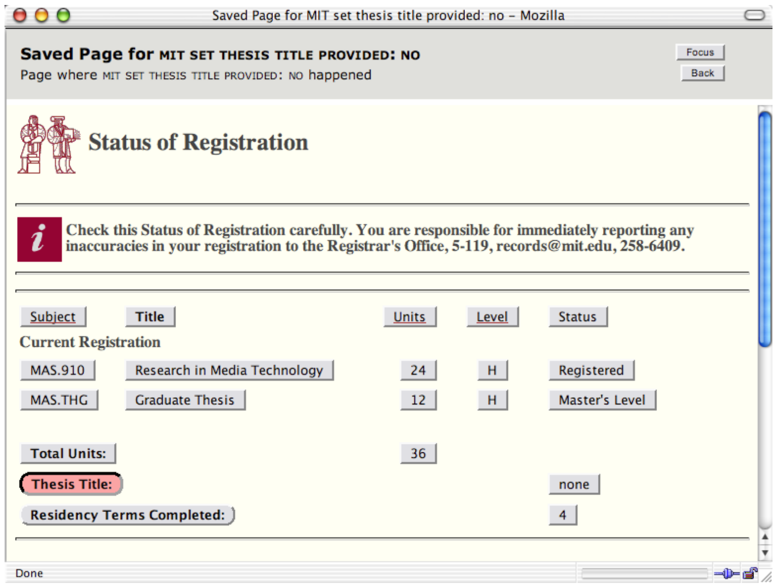
Figure 2-33: Viewing why the thesis title was unsuccessfully provided and the unsuccessful step

and approved by the department, but that there is a problem with his thesis title and that was before the thesis requirement check completed, so he marks everything else correct. Woodstein marks the thesis title check incorrect and the process of setting the thesis title as also incorrect (Figure 2-33).

The student selects the thesis title set process and sees its page context (Figure 2-34). It does look like his Institute is at fault, so he decides to complain. He goes over to the debugging trail view and clicks "Complain". Woodstein automatically generates an email with his debugging path that he goes ahead and sends (Figure 2-35).

2.9.1 Summary

The user applied a top-down debugging strategy and, in this case as in the previous one, Woodstein guided the user through identifying the incorrect looking data, and narrowing down the exact rep that was the problem. In this example, unlike the other, the user was correct and the web site was wrong so Woodstein enabled the user to easily generate a complaint about it.


Saved Page for MIT set thesis title provided: no – Mozilla

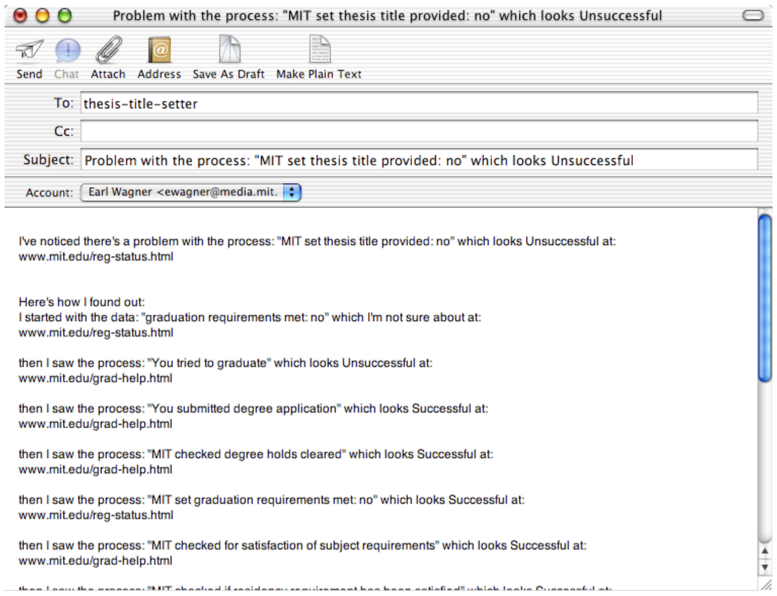
Saved Page for MIT SET THESIS TITLE PROVIDED: NO
Page where MIT SET THESIS TITLE PROVIDED: NO happened

Status of Registration

Check this Status of Registration carefully. You are responsible for immediately reporting any inaccuracies in your registration to the Registrar's Office, 5-119, records@mit.edu, 258-6409.

Subject	Title	Units	Level	Status
Current Registration				
MAS.910	Research in Media Technology	24	H	Registered
MAS.THG	Graduate Thesis	12	H	Master's Level
Total Units:		36		
Thesis Title:				none
Residency Terms Completed:				4

Figure 2-34: Viewing the saved page for the unsuccessful setting of the thesis title


Problem with the process: "MIT set thesis title provided: no" which looks Unsuccessful

To: thesis-title-setter
Cc:
Subject: Problem with the process: "MIT set thesis title provided: no" which looks Unsuccessful
Account: Earl Wagner <ewagner@media.mit.

I've noticed there's a problem with the process: "MIT set thesis title provided: no" which looks Unsuccessful at:
www.mit.edu/reg-status.html

Here's how I found out:
I started with the data: "graduation requirements met: no" which I'm not sure about at:
www.mit.edu/reg-status.html

then I saw the process: "You tried to graduate" which looks Unsuccessful at:
www.mit.edu/grad-help.html

then I saw the process: "You submitted degree application" which looks Successful at:
www.mit.edu/grad-help.html

then I saw the process: "MIT checked degree holds cleared" which looks Successful at:
www.mit.edu/grad-help.html

then I saw the process: "MIT set graduation requirements met: no" which looks Successful at:
www.mit.edu/reg-status.html

then I saw the process: "MIT checked for satisfaction of subject requirements" which looks Successful at:
www.mit.edu/grad-help.html

Figure 2-35: Complaining via and automatically generated email

Chapter 3

Background and Related Work

3.1 Context-Aware Computing and Software Agents

As people use computer systems, they often repeat the same task, such as checking a web page to see an account balance. Other times, they perform new tasks that are conceptually related to previous ones, such as checking a shipment tracking page after ordering an item. The systems people use today typically provide the same interface to the user regardless of the user's history. Though recent innovation has resulted in web browsers that fill in urls and personal data, the web browser itself doesn't know to go out and retrieve a web page on its own, and users today have no easy way of telling it to. A web browser, like most other software applications, is modeled on a tool that behaves predictably but not intelligently.

Context-aware computing offers alternate vision in which systems can adapt, evolving to better serve the user[32]. This research emphasizes how a system, guided by its history of interactions with the user, can act more proactively to provide better service. Software agents are a useful way of framing this proactive behavior by a system. They serve as an important means for adaptation by monitoring what a user is doing and giving help or other assistance[28]. Unlike applications today which feature tool-like interfaces, agents are modelled on human agents and act autonomously for the user's convenience. Just as we depend upon others for services like making arrangements like for travel, or preparing documents like our taxes, we may depend upon agents to perform activities on the computer that we find tedious, burdensome, or confusing. Just as we interact with people by saying things like "what's happening now?" rather than "file-¿open-¿my_doc.doc", we can experience more human-like interactions with an agent-based user interface.

As we saw in the examples of the previous chapter, Woodstein is an agent that works in the user interface. It answers questions about data and processes with information it has collected both by watching the user and retrieving relevant pages. It is modelled on reporters who know to look for the answers to “who?”, “what?”, “when?”, “where?”, “why?” and “how?” in assembling the facts of an event and writing a story. Indeed, its name is inspired by actual reporters[1]. A reporter figures out what’s happening by interviewing people chasing leads and putting together the big picture. Similarly, Woodstein recognizes what the user’s doing and simulates and retrieves information about what the web sites are doing to put together a complete, integrated account of what is happening. It helps the user “follow the money” to see exactly where some data, even a quantity of money, came from.

In the next chapter, we will look more closely at exactly how Woodstein was designed and implemented. Through the rest of this chapter, we will see related research that has influenced its design and implementation.

3.2 Interaction with Agents through Plan Recognition

3.2.1 Interactions with Agents as a Discourse

The closest work to this in interface style and spirit is Collagen [39]. Collagen works to guide users through tasks, such as making flight reservations or setting up a timed recording on a VCR. It tracks the steps of the user’s activity, and matches them through plan recognition to discourse models. Collagen differs from Woodstein in that it focuses on guiding the user through the normal operation of a system. Woodstein’s guidance, however, is intended for helping the user understand what has happened when something goes wrong.

Like Woodstein, Collagen is proactive. It typically features a embodied social agent in the interface, often rendered as a face. The agent, represented as a social actor in the interface, can take action directly when authorized by the user. Woodstein, on the other hand, records a user’s actions and web site reactions, but only presents the records in its the interface.

[1] “Woodstein” is the nickname Washington Post Editor Benjamin Bradlee used for Bob Woodward and Carl Bernstein who discovered President Richard Nixon’s involvement in the Watergate burglary. When Bradlee wanted to know what Woodward and Bernstein were working on and had discovered, he’d yell out to the newsroom “Woodstein!” to bring them in and update him.[5]

Both Collagen and Woodstein are capable of recognizing commonly occurring high-level goals by the concrete steps that the user is carrying out, though this is an area where Collagen is much more sophisticated. Woodstein is intended to do some of the things Collage does but in a new domain, e-commerce processes on the web.

Collagen is inspired by the SharedPlan[19] model of discourse in which the topics being discussed may be related hierarchically and the objects referred to by conversation participants exist in a shared space. For instance, one system built using Collagen helps with planning an airline route. It features an agent that offers suggestions to the user by referring to the users queries, and can also refine queries making them more or less specific in order to find an appropriate number of results. In this approach, the agent and user interact to satisfy the goal of finding a good route.

3.2.2 Plan Recognition

Plan recognition is used to extrapolate the next action of an agent, or infer its goals, based on its actions[8]. In early work on plan recognition, Cohen, Perrault and Allen [9] distinguished "keyhole" plan recognition, in which the observer received no cooperation from the agent, from "intended" plan recognition in which the agent is cooperating with the observer to make its goals known. User actions may also be interleaved, in which the time extent of multiple plans may overlap. Interleaved plans are difficult to recognize, however, since it's unclear which plan is active for a given action.

Plan recognizers typically accept process models that are partial orderings; A has to happen before B or C, but the order of B and C often doesn't matter. Less powerful plan recognizers require the steps of a process to be totally ordered, in which when each step happens in relation to other steps is specified.

When a user's plans are not interleaved, and the steps of the process models are totally ordered, parsing a user's plan is isomorphic to parsing strings. With these constraints, specifying user plans creates a formal grammar and all possible user actions are a formal language. Recognizing and parsing a plan hierarchy from a user's actions involves "parsing" a formal language in which the tokens are the user's actions. Just as a string of source code or natural language is parsed to create a parse tree, the sequence of user actions is parsed to generate a history of a process.

Since the intention of Woodstein is demonstrate a new application of plan recognition and not necessarily advance the state of plan recognition, it features a simple keyhole plan recognizer. User plans are recognized with a string parser that has the two constraints mentioned above - they cannot be interleaved and process models must be totally ordered.

3.3 Inspecting Web Pages

3.3.1 Analysis of Web Pages

A web page in HTML is usually designed for appearance; its dominant primary structure is its formatting. This has posed challenges for automatically analyzing pages, in order to extract and aggregate information from many web pages, for instance. This is useful in, for instance, finding the lowest price for an item among many retail sites. To support increased semantic structuring of documents on the web, the web standards community has been promoting XML as a standard format for annotating data in web pages to directly support extracting data.

Analyzing HTML web pages has been the focus of web information extraction[26]. Some research has focused on developing algorithms that can generate structured data from the way data is formatted in a page. Further research has examined how to support easily adding descriptions of data formatting in a page to support processing.

However, the issue of collecting and analyzing a user's personal data has not been an area of focus in web information extraction. Showing the semantic connections among the data in the pages a user interacts with is even farther outside the focus of this area. For instance, the charged amount on a user's credit card may be the same as the total purchase price. If it is, it can be found by matching the numbers. It's more important for the user to know when it is not, however, and matching won't discover that. Instead, a model of the purchase process is required to guide matching the entire transaction.

Woodstein is not intended to advance the state of research in this area but instead to demonstrate a new application for extracting data in a user's web pages to show the semantic connections among the user's data. To do this, it requires either that the pages themselves are already annotated, as in XML, or it accepts page models to guide analysis of pages. It features a simple page analyzer for extracting data and annotating unannotated pages.

3.3.2 Autonomous Agents that find Contextually Related Pages

The creation of the Web coincided with a growing appreciation of software agents for personalized retrieval and presentation of information[33]. Since that time, much research has focused on autonomously retrieving information useful to the user from the web. In particular, some projects have focused on allowing the user to find information contextually related to the data in a page, including Letizia[27], Watson[7] and Margin Notes[38]. These systems search other documents to find ones related to the current document. Unlike these approaches which typically use keywords or modification times, Woodstein is guided by its record of the user's process to retrieve semantically related pages.

3.3.3 Web Page Transduction and Annotation

The idea of supporting the direct inspection of web pages grew out of annotating pages as they're being transcoded. Web page transcoding is the process of modifying web pages between when they're sent by a server and when they're received by a browser. Transcoding has traditionally been seen as a way of altering the appearance of web pages intended for desktop PCs so they can be rendered on palm computers, or translating pages into other languages. Like many systems for transcoding, Woodstein is implemented as a web proxy. It monitors the browser's requests and monitors and annotates pages returned from the server.

Some projects have looked at transcoding more broadly for other applications. Unlike traditional transcoding in which a fixed transformation is applied to each transcoded page, the WBI project supports transcoding based on dynamic and personalized data[4] [3]. One plugin for the system annotates each link in a page with a traffic light icon indicating the link's ping latency, a measure of how responsive the referred to server is. Another plugin tracks a user's browsing history and allows the user to search through that history. These plugins demonstrate WBI's support for collecting information to guide the annotation of the page. Annotations may add extra controls and behaviors not originally included by the web page designer.

A similar approach supporting inspection of data in a page is taken by Domingue et. al.[16] with Magpie. In that system, an extension to the web browser annotates recognized data items and provides a menu allowing the user to ask about the data. For instance a name on a web page can be clicked to find the person's home page.

Woodstein builds on these approaches by personalizing pages even further. It allows the user to inspect the page data created by a process to access a record of the process itself. As we saw, a charge on the user's credit card can be inspected to access the record of the overall purchase process.

3.3.4 "Recontextualizing" Data

Woodstein is also influenced by a vision of hypertext different from what the web offers today. Ted Nelson has long been an outspoken proponent for stronger connections among linked material. His project Xanadu is a well known attempt to provide technological support and structure for sampling copyrighted works within other copyrighted works, while retaining links to the originals. Another of his projects, ZigZag[34], though less well known, is more similar in spirit to Woodstein. It supports relationships among data through "zipper lists". Each zipper list is a container or category for some of the data tracked by the system. Zipper lists are connected "sideways", with the same data in one zipper list appearing differently in others.

From this perspective, Woodstein manages three zipper lists, one corresponding to each of its views. It tracks where a process or data item rep fits within the process history, data history and where it appears within a web page. The rep is inspected in one view to reveal its role in other views.

Of course one of the key factors in the success of the web is its simplicity, that any web page can feature links to any other and conversely that data can be decontextualized and not linked to anything else. Since anything can link to anything else, links carry no semantic structure with them, however, and eliminating the context of data often prevents engaging with them more richly. A user can download a spreadsheet with her transactions from an online bank, but how can she see larger trends in the origins of those transactions or even just inspect one more closely? Each transaction has been isolated to include a few data items including an amount and relevant dates, then detached from its history, or decontextualized. Woodstein restores the semantic relationships among data to, in effect, "recontextualize" the data and support easy exploration of their history.

3.4 Inspecting and Learning about Systems

3.4.1 Constructionism

Seymour Papert, in his book "Mindstorms"[35], describes his fascination as a child with the gears in cars. He would look inside and think about how the gears interacted to cause the wheels to turn. Later, with algebra, he discovered a field that described the ideas he had already grown familiar with. He describes systems like the gear assemblies as "objects to think with" that in thinking about, cause an observer to see more abstract relations and concepts more clearly.

Constructionism is the idea in the field of learning that a person learning about something doesn't just receive knowledge from books or teachers, but must construct her own understanding of the domain. A particularly effective way for a person to learn something is by by thinking about the ideas involved and creating artifacts that embody and externalize those ideas. The process of creating the artifact inspires the refinement and development of the learner's understanding as the ideas are made more precise and concrete. The artifact also provides others with an object they can interact with that acts as a record and conduit for the ideas, perhaps inspiring other ideas and thus acting as "objects to think with".

In "Mindstorms", Papert describes using computers to create systems known as "microworlds" representing domains such as math or physics. He describes one system in particular, the Logo programming language, in which geometric relationships and behaviors could be programmed. For instance, a plotter on paper or cursor on the screen acting as a "turtle" could be instructed to draw out polygons. Learners themselves would create programs to see geometric ideas in different ways. Others could then take those programs and modify or extend them, to go on and see the relationships of the domain in still more ways.

3.4.2 "Turtles All the Way Down"

The Logo community uses the expression "turtles all the way down"[2] to describe systems in which new abstractions are added to the minimal language core in layers. With well-designed core functionality, the layers can all be implemented using the same techniques, making them easier to understand. This idea is also promoted by the Smalltalk[18] environment, in which all values, even integers and boolean values, are objects. New abstractions can be built taking the features of an object for granted and leveraging the abstractions it provides without requiring that handling for special cases be added to the language or environment. Thus there are no conceptual barriers to understanding increasingly lower levels of the system, since its rules are uniform at every level of abstraction.

Woodstein uses a related approach to present the processes associated with a web page in a standard form. Inspecting a process or data item in the page, reveals the process and associated pages at a lower level via the same presentation, without the need for "special cases" in explanation.

3.4.3 Reflective User Interfaces

Although not commonplace today, some research systems have been developed to feature a "reflective user interface", also known as "reflective accounts"[17], or "implementational reflection"[37] as opposed to computational reflection. Computational reflection[41] is a feature of some computer systems in which a system allows its modules to inquiry about its capabilities and status at run-time. A reflective user interface extends this functionality to the interface, allowing a user to inquire about a system's status as it is running. This is useful, of course, only if the user can do something with this information, so reflective user interfaces have traditionally been limited to programming systems and knowledge-based (expert) systems, in which programmers and experts are able to modify how the system operates and instantly see the results. Some notable programming systems with reflective user interfaces include the Lisp Machine Operating System[45], Emacs[42], Smalltalk and Squeak[23], as well as some expert systems including Teiresias[15]. Each of these systems

[2]These turtles are different from a Logo turtle, which came from the size and speed of the original plotting device. This remark refers to a story told by the astrophysicist Stephen Hawking about the philosopher Bertrand Russell: "An elderly lady confronted Bertrand Russell at the end of his lecture on orbiting planets, saying, 'What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise.' Russell gave a superior smile before asking what the turtle was standing on. 'You're very clever young man, very clever,' replied the old woman, 'but it's turtles all the way down.'[20]

allows a user to ask about how some process is being performed, or some knowledge is being reasoned about, and permits the source code or rules for reasoning to be modified.

Woodstein creates a reflective user interface layer over pages on the web. It allows users to inspect data items and descriptions of processes to see the processes that created them. Unlike those systems however, Woodstein does not allow users to change how the processes they've initiated work. Instead, it is intended that the visualizations it provides be used to understand the processes and diagnose problems when they occur.

3.5 Techniques for Program Understanding

Tools for visualizing and understanding the dynamic properties of a system have traditionally been the focus on software visualization. The techniques developed with this field for simplifying the presentation of complex processes have not been widely adopted in the research and development of computer systems, though they can be invaluable for understanding the status of a system and how it works. These concepts have guided the design of Woodstein's views.

3.5.1 Program Activation Trees

Program activation trees are diagrams that show the sequence of steps in the history of a process [1]. They are generated from the run-time execution of the process. Unlike the program source code which indicates all possible control branches, activation trees only show the branches actually taken by execution, and only the functions actually called.

3.5.2 Program Slices

Program slices are akin to the idea in business of "audit trails". They capture the history of some data, showing all of the processing it has undergone. More specifically, program slicing [46] is a software engineering technique for focusing only on the parts of a program that affect the value of a particular variable. It is helpful for debugging, when a programmer knows a variable has the wrong value and wants to know how it was computed. Slicing can be performed statically by analyzing a program's source code to see all possible ways the value of a variable may be computed. A program may also be sliced dynamically [24], to find exactly how a variable was computed given the program inputs of a single execution.

Slicing, and dynamic slicing in particular, can be helpful for debugging by allowing the programmer to determine exactly how a variable resulted in having the incorrect value that it does. Woodstein generates dynamic program slices of a data items and invoked processes and visualizes these slices in the data history view.

Program slice tools typically highlight the lines of code, modules or files in a slice [2]. This is useful for programmers, for whom the source code is the primary representation of the program. Within the domain of web actions, however, we don't expect the abstract models to be particularly meaningful to end-users. Rather than presenting the abstract description of the process, Woodstein generates explanations of the process' actual concrete execution.

Some tools present slices via control-flow graphs or program dependency graphs[21]. Woodstein presents the program dependencies in the data history view, and the program execution tree in the process history view.

3.6 Debugging

Debugging is an underappreciated problem in computer science[29]. Despite studies of programmers that show that half of the time of software development is spent on debugging activities[10], there is little work in computer science that explicitly seeks to make debugging easier and more effective. Tools available in today's commercial programming environments are essentially unchanged from those that appeared in programming environments thirty years ago: function trace, breakpoints, line-by-line stepping. There is, of course, nothing wrong with these features; studies have observed that novice programmers can fix 80% of their bugs just by stepping through the executing code[6]. However, there is more to debugging than just stepping.

Some work has been aimed at understanding the cognitive processes involved in debugging. Conventional tools don't explicitly support the cognitive processes of debugging, which include:

Visualization "What's happening?" "How do I get an overall feel for what's going on?"

Localization "Where is the problem?" "What specific step or piece of data is responsible?"

Instrumentation "What's the history of this event or data item?" "Did this happen?"

Repair "What are the consequences of a proposed fix?"

We will refer to debugging as involving all of these steps, while diagnosis of bugs involves only the first three.

These insights have guided the development of tools that help programmers ask and answer questions about the dynamic behavior of systems.

3.6.1 Tools for Debugging

This gap between current tools and the tasks involved in debugging is largely a user interface problem[43]. ZStep[31] is one of a series of debuggers built to support these cognitive processes, especially visualization and localization. ZStep is a reversible debugger - it records every step of the execution of a program and allows the programmer to run the program backward as well as forward.

Some of ZStep's innovations include:

- A complete history of program execution and output
- Controls to run the program forward and backward
- Graphical output is associated with points in the program history, allowing it too to be run forward and backward
- One-click access from graphical objects to the code that drew them and vice versa
- An inspector that displays run-time values for selected source code
- Incremental control of the level of code detail displayed

Woodstein builds on these features and extends them to the domain of web processes. Like ZStep, Woodstein records a complete history of a process and allows the user to jump back to any point of that history or replay it forward or backward. While ZStep tracks the graphical output of the program, Woodstein tracks the pages generated by the process. The user can go from a data item in a page to the point in the process in which it was set, and vice versa.

3.6.2 Level of Detail Control

Like ZStep, Woodstein allows the user to inspect the process at varying levels of detail. The user is able to see her exact interactions with a page including the links clicked and the text entered. An interaction can be replayed using the stepper. Though there exist some tools for recording interactions with web pages, such as Tower's WebCapture[22] and LiveAgent[25], these don't provide any additional structure beyond the linear sequence of web pages.

It's important that debugging tools show only what's relevant at any given moment and allow the user to control how more detail is progressively revealed. Too often, computer systems show too much information, making it difficult for the user to become oriented and to know what to focus on. A benefit of Woodstein's process history view over a linear history, for instance, is that the user can see at a high level which processes need further investigation. She can drill-down in only those that look incorrect, ignoring the ones that look correct. Similarly, in a data history view, it would be overwhelming to see all possible values that contributed to the computation of some data. Instead, Woodstein shows only the most recent layer of inputs, then allows those to be inspected.

3.6.3 Debugging Styles: Top-Down and Bottom-Up

Tools that only record the sequence of pages a user interacts with are only able to provide a linear history and don't support more sophisticated diagnostic strategies. By recording and presenting the hierarchical structure of a user's action, Woodstein enables the user use a "top-down" approach of "drilling-down" through levels to find what went wrong. It also allows a user to start from some data that appears incorrect and use a "bottom-up" approach of seeing how that data was computed and what the faulty processes were.

A user doesn't have to use one strategy over another; rather they both can be effective and a bug is often found by using a hybrid approach. For instance, a user might see that some data looks wrong, use a bottom-up approach by tracing it back to the process that set it, then use a top-down approach to see why the process occurred.

3.6.4 Programmer Hypotheses in Debugging

In the process of debugging, a programmer notices that the program emits incorrect data, or runs a process incorrectly. She then learns the dynamic behavior of the program well enough to identify the bug, the origin of the incorrect data or the invocation of the incorrect process. As she learns about the state of the program, she keeps track of the data and processes that are known to be correct, those that are known to be incorrect and those whose correctness is unknown. Taking into account the causal relationships among the data and processes, she formulates hypotheses about the correctness of the program's behavior and uses these hypotheses to guide her diagnosis until the bug is found.

3.6.5 Agent Support for Debugging

Traditional debuggers present data but don't provide explicit support for programmers' hypotheses. They allow inspection of the history of data and processes, but don't keep track of the programmer's current view of their correctness. ZStep, like other debuggers, has a tool-like interface and provides views of a process only on the demand of the user. Beyond tools, programmers and users can benefit from agents that play a more active role in eliciting, managing and helping users share their hypotheses. This is an area in which Woodstein goes beyond ZStep as well as all other known debuggers.

The need for an agent to assist in diagnosing and resolving bugs is particularly acute for non-programmers, who don't typically have the experience that programmers do in reasoning about formal processes. Some researchers have focused on how to better support end-users and novice programmers in debugging. Margaret Burnett, in particular, has worked on enabling end-users to debug spreadsheets within the area of "end-user software engineering" [47] [40].

As a user works to diagnose a problem, it is helpful to have an agent that keeps track of possibilities she's checked and recommends what should be checked next. Woodstein provides help in this way and supports the user by automatically performing the process of elimination, either in a top-down or bottom-up style, to identify the faulty process or data item.

n

Chapter 4

Woodstein's Design and Implementation

We have seen how Woodstein's interface works and the research and products that have influenced it. In this chapter, we will discuss its architecture and the design decisions involved in implementing it. We will look at how Woodstein's features and interface are implemented as components and how all of its components work together to create its functionality and interface.

4.1 Design Goals for Woodstein

Woodstein occupies a point in a spectrum of possible designs. As we saw in the previous chapter, it builds upon work in many different areas, such as plan recognition and web information extraction. Rather than trying to improve upon the work in each of these areas, we sought to develop a new application of their results.

Woodstein was designed to help users in debugging problems by themselves. Thus we've focused on how to present web actions that may be extended in time, or span multiple sites. To a lesser extent, we've also considered on other goals, such as helping users avoid problems in the first place by guiding them through processes. We also sought to make the results of a user's investigation useful to others, including customer support representatives. However, there is much more work in these areas to do, as we will see in the future directions in the last chapter. In this section we will see the main goals for the design of Woodstein.

4.1.1 All Data and Processes are Inspectable

It's important that user be able to see and interact with information in its original context. Inspecting information should reveal its historical context including the history of how it was computed and how it got there. With Woodstein, the user is able to directly inspect data and processes in the pages within which they appear. Similarly, when interacting with data and processes in the agent's views, their original contexts are immediately accessible.

4.1.2 Support User Reasoning about Hypothetical Possibilities

We wanted to help users in reasoning about why something did or did not happen. Users should be able to point to some process and ask why it happened, or some data and ask why it was computed to have the value that it did. Beyond that, however, the user should be able to ask why something did not happen. Woodstein supports the user in investigating these possibilities, as the user in the purchase example was able to see why the item had not arrived yet. However, supporting questions about hypothetical possibilities remains an area for future work, as we will discuss in the last chapter.

4.1.3 Effective Visualizations for End-Users

Woodstein's visualizations are intended for end-users allowing them to see the underlying connections between information in pages. Although they would feature concepts borrowed from programming, the amount of learning required to use the system should be minimal. In particular, all concepts should be as concrete as possible and all abstractions should be expressed in terms understood by the user. Processes and data should always be described in an intelligible way, and the visualizations themselves should be accessed via questions in terms that users understand.

In addition to being understandable, we also wanted the visualizations to help the user efficiently diagnose the sources of problems that arise. In particular, the user shouldn't have to revisit every step of an action to get a sense of whether it is correct. With Woodstein, at any level in the process history, the user is able to see the result of each process to determine whether it was successful. In the graduation scenario, it is helpful to the user to be able to instantly see the categories of requirements that weren't satisfied, then which requirement in particular wasn't satisfied.

4.1.4 Fail-Soft Plan Recognition and Page Analysis

The agent should track all of a user's actions online. It should provide help, even within a spectrum of circumstances:

no cooperation from the web site in which case at the very least, the user has a saved copy of his pages

process descriptions from the web site perhaps in a web-services or business process language, would allow the agent to recognize a user's action and web site reactions

process descriptions from the web site and annotated pages as envisioned for the Semantic Web, providing the best support

The development of Woodstein has focused on demonstrating the potential offered by the last possibility, though it supports the others.

4.1.5 Help Customer Support

A final goal was to provide help to customer support. Rather than have the user describe the steps he's taken in investigating a problem, he could send a record accumulated by the agent. This remains underdeveloped, and we discuss further possibilities in the the future directions chapter.

4.2 How Woodstein is Implemented

Over the rest of this chapter we will see how Woodstein works internally by revisiting some of the steps in the examples from the last chapter. We will see how Woodstein:

works with the user's browser as a web proxy to track a user's steps and pages

recognizes a user's goal by recognizing plans from a library

represents a user's data and processes by instantiating processes from process models

simulates and verifies web site reactions and user data by acting autonomously as an agent to retrieve related pages

manages views of process and data history from its process and data representations

manages user annotations and hypotheses by performing the process of elimination

Woodstein's overall architecture includes an action analyzer, a page analyzer and an information retriever, as shown in Figure 4-1. In the figure, components are shown as boxes and the flow of data among them are indicated by arrows.

4.3 How Woodstein Works with the User's Browser

Woodstein works with the user's browser to generate an enhanced "clickstream", featuring not only the pages the user loads, but also the user's exact clicks and typing from interacting with a web page. With this record of the user's steps, it goes on to build a history of the user's action, which we will see in a later section. For now we will just focus on how it gets this record.

4.3.1 Types of User Actions

Woodstein creates events when a user:

- loads a page
- clicks on a link or control
- types in a text field

It does not recognize other events, such as when the user moves the mouse or types a new url into the location bar before loading a page.

The events Woodstein creates are added to its record of all user events and passed onto the plan recognizer.

4.3.2 Woodstein as a Web Proxy

Woodstein is a web proxy. It stands between the user's browser and the web, receiving all requests for web pages from the browser and all pages from the web meant for the browser. From this position, it monitors the user's page loads and analyzes, annotates and saves pages before they reach the browser.

Woodstein creates a page-load event when it gets a request for a page from the browser. It creates click and type events when the user interacts with controls in the page. To receive

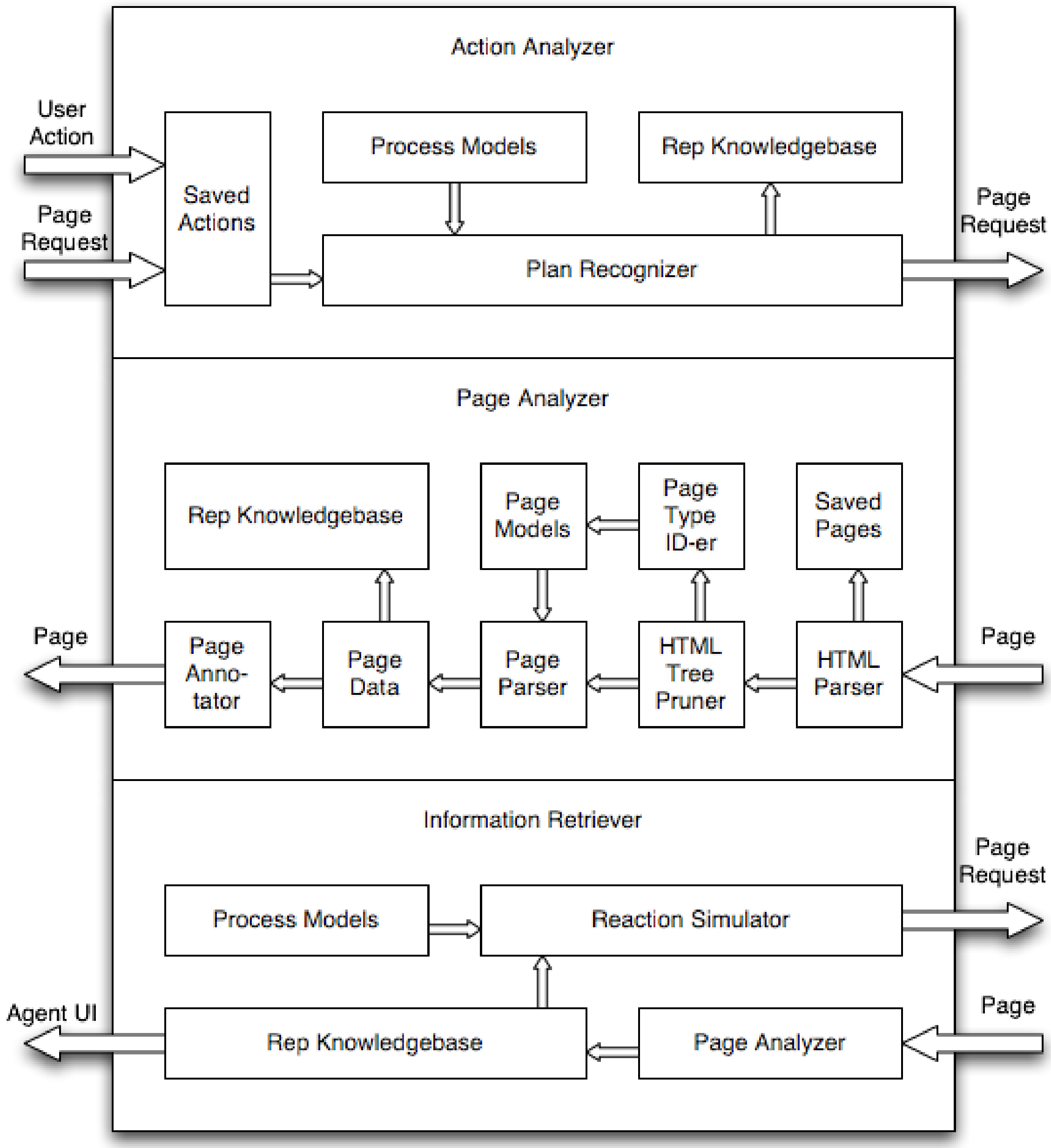


Figure 4-1: Woodstein's Architecture

these events, it annotates all controls it finds in a page and adds the Woodstein icon as a floating watermark before it passes the page on to the browser. Then, when the user clicks on a link, for example, the annotated control sends a message back to the proxy through an http "get" request that passes along event information while reloading the watermark image. As we will see, many messages are sent by the interface through these get requests that just reload watermark image while sending information back to the proxy.

Comparison of a Web Proxy with a Browser Module

We also considered implementing Woodstein as a browser module. An important feature of a web proxy is its flexibility; the agent can reside on the user's computer, or on a server. It may even be relocated from one computer to another. However, a module offers an important advantage over the proxy by being able to access the actual HTML DOM (Document Object Model) tree rendered by the browser. This is useful when the code for the rendered page is different from the static HTML code. Many web pages perform some JavaScript processing of a tree before it is rendered and some pages perform substantial processing of the tree. Unlike a browser module, a web proxy is only able to access the static HTML for a page before it is passed on to the browser[1]

"On the web nobody knows you're written in Lisp."[2]

For a deployed product then, a web browser module may be the better choice. In implementing a research prototype, however, we were also guided by our expectations for the development process. We anticipated needing the ability to do extensive rapid prototyping, as well as symbolic artificial-intelligence-style reasoning. We chose to implement Woodstein as a proxy in Lisp because of these reasons, as well as because of our familiarity with the language and its use in web programming. In fact, the short time between editing a function for generating a view's display, reloading the function into the running Lisp system, and reloading the view page (on the order of 5 seconds) allowed for many, many more iterations during development than would be possible using a traditional language like Java or C++ which would require the slow recompilation of an entire module. Even major modifications

[1]One newly standardized type of DOM event, a "mutation" event is created whenever the tree is modified. An mutation event could be used to notify the proxy of changes to the tree, although then coordination between the page and proxy becomes more challenging and error-prone

[2]The unofficial motto of the 2002 International Lisp Conference, according to Fred Gilham in the Usenet post with message-id: "u7lm4d3vl7.fsf@snapdragon.csl.sri.com"

to the architecture for representation could be recompiled and the problem scenario re-run within 20-30 seconds. The extensive use of JavaScript for the interface also allowed new ideas to be implemented quickly, although the immaturity of debugging tools for JavaScript still made developing the interface behavior substantially slower than developing the core agent.

4.3.3 How Woodstein Recognizes and Annotates Pages

Woodstein analyzes and annotates a page in three stages. First, it analyzes a page to recognize its role in the process, such as "order confirmation page". Then it recognizes all the data in a page and compares the page data with data it is tracking. Finally, before the page is passed on to the browser, it annotates all controls, creates hidden buttons for data and process in the page and adds the watermark logo.

4.3.4 Recognizing Page Roles

Woodstein recognizes a page's role in the process either by matching the url, or by looking for particular data or processes in the page. For instance, an order confirmation number for a purchase must appear in the order confirmation page.

4.3.5 Extracting Information from Web Pages

Woodstein can either accept pre-analyzed pages or it can analyze pages itself using separate page models. Pages may be provided by the web site with the data and processes already surrounded by specific HTML "span" tags. Or Woodstein can perform the page analysis. It features a simple analyzer that tokenizes the HTML page and parses it using a string parser and a grammar for the particular page. It looks for data and their labels recognizing, for instance, "total price" in bold as a data item label, and the price that follows as the "total-price" data item.

4.3.6 Annotating Pages

In Woodstein's last pass analyzing a page, it performs the final annotations of reps before the page is passed onto the browser.

Page controls, including links, form buttons and text areas, are instrumented with JavaScript code to notify the agent when the user interacts with them. When a user clicks on a button, for instance, Woodstein receives a message from the annotated page in the web browser and updates its history of actions performed by the user.

Buttons are created for the data and process reps in the page. These buttons are “incognito” and hidden when the inspector isn’t turned on.

A separate stage is necessary because, for instance, there may be buttons for processes that hadn’t happened when the page was first retrieved, but have since occurred. In the order scenario, when the user views the help page for placing the order, some of the actions haven’t hadn’t yet happened. Those events should become buttons when the user loads the same saved page for placing the order after having performed the remaining actions, and thus the annotations of the page should match the current state of the user’s action.

Finally, Woodstein’s logo is added as a watermark to the bottom-right of the page indicating that Woodstein is monitoring this page. Clicking on this logo turns on the inspector and reveals the annotated data and processes in the page.

4.4 How Woodstein Recognizes a User’s Goal

As the user performs the steps of an action, the web site reacts to the user’s steps and the user’s overall plan emerges. We will now look at how Woodstein recognizes the user’s goals from his sequence of steps. First we will see how Woodstein represents user plans as they are being performed. Then we will see the algorithm that it uses to match a user’s steps with a plan from its plan library. Later we will see how Woodstein reasons about the reactions of the web sites affected by the user’s action.

4.4.1 How User Plans Are Structured

Although we all have a sense that we do things because we have goals, and satisfying those goals requires satisfying sub-goals, it isn’t necessarily obvious how that can be used to analyze an interaction with a website.

In approaching this, we took inspiration from Collagen, which represents a user’s interaction with an agent as a collaborative discourse. Collagen manages a user’s plan tree consisting of goals and sub-goals and provides an interface agent which guides the user in reaching a goal it shares with the user, such as programming a VCR. Woodstein, on the other hand, just parses a user’s steps to generate a plan tree, using the tree to infer the reactions of the web site and visualizing it as a record of the user's action.

Woodstein models the interaction between the user and the web site, though for typical web sites, this isn't a collaboration. Instead, the user interacts with the web site in doing an action. The web site performs its own reaction to the user actions, which in turn causes the user to perform further actions. Woodstein tracks this interaction, then visualizes aspects of it when needed by the user.

Woodstein represents a user's plan at increasingly specific levels. At the top level are processes that it infers are happening based on both other processes it has inferred are happening and the actions of the user and reactions of the web sites. At the bottom level are the steps involving the user and web sites. Here are the types of processes with examples:

Abstract Processes abstract tasks that Woodstein recognizes, such as placing an order.

Steps actions that are atomic from the perspective of the interface.

- **User Actions** steps such as loading a page, clicking on a button or typing in a text field.
- **Web Site Reactions** actions such as updating a shopping cart in response to the user adding an item, or accepting the user's choice of payment method.

Abstract processes are trees with steps as the leaves. An step is either a user action, or a web site reaction. User actions cause web site reactions. Woodstein directly sees only the user actions and must be guided by its process model to look for web site reactions and load the pages it expects to be updated.

Another issue arises in determining how to manage data resulting from the interaction between the user and a web site. The user supplies some data, by clicking to indicate choices such as an order's shipper, or typing to enter information such as an address. On the other hand, a web site manages the vast majority of the user's data, by taking the user's input and processing it, creating a shopping cart, a subtotal, an order address, an order and so on. All of this data arises from the user's interactions with the web site, and one important role of Woodstein is to represent the web site's processing of data, and allow the user to diagnose errors in the results, for instance if the web site adds the incorrect number of items to the shopping cart. This processing is visible to the user and Woodstein only as data that appears on web pages.

Since the results of web site reactions are only accessible as data within web pages, we infer that web site reactions have happened by looking for the data we expect them to set. This is a good fit for most cases, such as when a web site sets the shopping cart contents after a user adds an item. The fit is less good when actions are implicit in the pages the user loads. When a user adds the item, the name of the item must be created. Before the user loaded the page, the web site set and created the data associated with the page, such as the items it features or the default settings for controls. So even though these data creation actions are discovered after the page is retrieved, we put them before the user's page load in the process history. This approach also works well with the data history perspective, as we can see in the example involving the selection of a shipper. First the default shipper value exists. Then, the user approved it, clicking "ship", which caused Amazin.com to set the value of the order shipper to be the default shipper.

4.4.2 Woodstein's Process Models

In the examples we saw, Woodstein had a predefined model of how the process would occur. For instance, it has a model for a purchase that begins with browsing and selecting items and ends with entering order details and confirming. Woodstein instantiates processes from its library of process models as it recognizes the actions performed by the user. In this section we see look at the format for these models and see the information they contain. In the next section we will see how a process model is converted to a parser for parsing user actions and generating the resulting process.

Model Formats

Examples of models for processes and steps can be seen in Figures 4-2 and 4-3 show

Models for processes and steps begin with the meta-information used to generate explanations in the interface. The main meta-information is the "name" string with a sentence description of the process or step. It is important to note that the name is different from the symbol for the process in which words are joined by dashes. The name sentence is in the past tense, like the name for the process itself, and is converted to the present or future tense depending on whether the process or step has happened yet. To facilitate reconstituting the sentence in a different tense, the subject, verb and object are also specified and stored separately.

```
(def-process-model you-purchased-from-<retailer> ()
  (name "You purchased from <Retailer>")
  (subject "You")
  (verb purchased)
  (object "from <Retailer>")
  (is-top-process t)
  (source nil)
  (steps
    you-placed-order-with-<retailer>
    <retailer>-requested-payment-from-<bank>
    <shipper>-delivered-order))
```

Figure 4-2: The process for a user placing an order.

```
(def-event-model you-clicked-add-to-shopping-cart ()
  (name "You clicked 'Add to shopping cart'")
  (subject "You")
  (verb clicked)
  (object "'Add to shopping cart'")
  (source retailer-book-page)
  (step (you clicked add-to-shopping-cart))
  (meta-actions))

(def-event-model <retailer>-set-shopping-cart-item ()
  (name "<Retailer> set")
  (subject "<Retailer>")
  (verb set)
  (object "shopping cart item")
  (source nil)
  (step (<retailer> set shopping-cart-item))
  (meta-actions
    ws-set--shopping-cart-item))
```

Figure 4-3: The steps for a user adding an item to a shopping cart and the shopping cart updating.

```
(def-meta-action ws-set--shopping-cart-item ()
  (function-sym ws-set)
  (arg-forms (shopping-cart-item current-item-name))
  (source retailer-shopping-cart-page))

(def-meta-action ws-set--total-price ()
  (function-sym ws-set)
  (arg-forms (total-price (ws-eval + shopping-cart-subtotal shipping-charge)))
  (source nil))

(def-meta-action ws-set--order ()
  (function-sym ws-set)
  (arg-forms (order (ws-make-whole shopping-cart-item total-price shipping-address shipper payment-m
             retailer-order-confirmation-page))
  (source nil))
```

Figure 4-4: Some meta-actions involved in placing an order.

Process and step models also include a source page. For processes, this is the page in which the process is described. For steps, it's where the step happened.

The last slot for a process model is the sequence of sub-processes for the process. The last slot for a step model is the "meta-action" for the step, which Woodstein uses to update its own representation of what is happening. These meta-actions change Woodstein's representation of the current process, by setting variables such as the item the user is currently looking at, or the name of the retailer the user is browsing. We will look more closely at how Woodstein manages user data with meta-actions later. Some of the meta-actions involved in the purchase can be seen in Figure 4-4

Step Verbs

Unlike abstract processes, which can be described with any verb, steps are described in a more standard way with a limited set of verbs. The verbs for user actions are just the things the user does that Woodstein can recognize. The verbs for web site reactions all describe how the web site manages data that Woodstein represents.

The types of user actions and corresponding verbs are:

Saw when a user loads a web page

Clicked when a user clicks on a link or form button

Typed when a user types text into a form field

Left when a user leaves a control in its default setting before submitting the form it is contained within

Note that a "Left" step is inferred when the process specifies that some selection is made and the user leaves a control in its default value. We saw this in the purchase examples with the user leaving the default shipper selected in placing the order.

The verbs for web site reactions depend upon whether the data being acted upon is primitive or composite. They are:

Primitive Data

Set when the web site sets a data item such as the current item a user is looking at

Matched when the web site matches data in a page, or entered by the user, such as a user-name and password

Composite Data

Created when the web site creates a composite data item from other data items

Updated when the web site adds or changes one of the data items contained with this data item

Matched when the web site matches page data

4.4.3 Woodstein's Plan Recognizer

Woodstein features a simple plan recognizer that takes the sequence of user steps as an input and creates a tree for the process it corresponds to. It is intended to recognize only simple plans and requires that plans are not interleaved and have steps that are totally ordered. To do this, it uses a string parser for parsing the sequence of user actions.

4.4.4 Creating the Parser

When a process model is loaded, an attribute grammar for the process is created in extended Backus-Naur Form (EBNF)[3]. This grammar consists of a start symbol for the top-level process, such as making a purchase. Each process definition results in the creation of at

[3]EBNF differs from BNF by adding the regular expressions $*$, $+$, and ?. A process step $foo*$ stands for zero or more occurances of foo, while $foo+$ stands for one or more and $foo?$ stands for zero or one.

least one rule. A complete rule with all of the process steps is used to recognize a completed process, while the incomplete rules with only some of the process steps are used to recognize an ongoing process. First, one rule is created with the process symbol as the rule symbol and the steps of the rule as the right-hand-side (RHS) symbols. Then, incomplete rules are created with an "incomplete" process symbol and some of the end steps removed. For example, a process with 3 steps results in the generation of one complete rule with 3 rhs symbols, one incomplete rule with the first 2 rhs symbols and one incomplete rule with just the first rhs symbol.

Although all processes with steps create rules, only user actions derive terminals. Web site reactions don't derive terminals (thy're e-rules with an empty RHS) and are automatically created as necessary.

4.4.5 Parsing User Actions

Woodstein maintains a representation of the history of the current process as a parse tree. When the user performs an action, it creates tokens for the action and passes the parse tree followed by the tokens as the input to the incremental parser. The parser reparses the tree, breaking some of it down if necessary, and rebuilds it to include both the user's action and the web site reaction nodes. Each node features some data known in parsing as an attribute, in this case, a process rep. As the hierarchy is created, when a node is created with other nodes as its children, the process rep is instantiated with the nodes' processes as children. So when the parser sees a sequence of nonterminal nodes with the symbols: "you-placed-order-with-<retailer>", "<retailer>-requested-payment-from-<bank>", "<shipper>-delivered-order", it reduces them to a nonterminal node with the symbol "you-purchased-from-<retailer>" and create a new "you-purchased-from-<retailer>" process rep for the new node with the children nodes' processes as children.

4.5 How Woodstein Represents a User's Data and Processes

In the previous section we saw how Woodstein takes the user's actions and matches them with its process models to recognize a user's plan. To do that, it instantiates its model process reps and creates actual process reps. In this section we will pause and look more closely at the process and data reps that it manages. In the next section, we will continue following Woodstein's operation and see how it takes the process hierarchy it recognizes and simulates web site reactions in response to user actions.

4.5.1 Woodstein's Representation of User Data and Processes

Woodstein creates a new representation, or "rep" object (see Figure 4-5)[4], for processes that it instantiates and user data that it tracks.

All reps have names used to describe them. Data reps receive their names when they are created and set, while process reps inherit their names from their model process reps. Reps have other information, in addition to their names. When a process instantiated, it has a parent process it is a part of, if it is not the top process, and it has children, if it is not an action. Other information is required by the particular views and we will see it in more detail when we see how the views are constructed. The process views require that each process have a result if it has completed or a status if it is ongoing. The data history view requires that a rep tracks the reps used to compute it and the reps it was used to compute. The saved page view requires that all reps refer to the saved page in which they first appear. Finally, the debugger view requires an email address to send a user's complaint to if the rep is found to be incorrect.

Managing reps within Woodstein's knowledgebase is greatly simplified by having a global namespace and no more than one rep of each type. For example, the user can only engage in one "you-purchased-from- <retailer>" action at a time since that symbol refers to at most one process rep. This simplification imposes a serious constraint on the flexibility of process models, however - a user cannot add two items to the shopping cart because there is only one "you-clicked-add-to-shopping-cart" process. Due to its magnitude, this limitation will be removed in the next version of the agent.

4.5.2 Woodstein's Representation of User Data

Woodstein represents user data managed by the web sites the user interacts with. For instance, as a user browses a vendor's site, Woodstein keeps a record of the current contents of the shopping cart and a history of how items were added so that the user can later revisit these steps. Woodstein updates this record as part of simulating web site reactions to user

[4]def-class is a macro that expands into define-class with slots having accessors of the same name and nil init forms

```
(def-class rep-class ()
  (id)
  (sym)
  (source-page)
  (owner-email)
  ;; parent and child relns for steps
  ;; shown in process history tree view (why)
  (step-parent)
  (steps)
  ;; parent and child relns for uses
  ;; shown in data history graph view (how)
  (uses-children)
  (uses-parents))

(def-class process-rep-class (rep-class)
  (model-rep)
  (tense)
  ;; the var set by the last meta action of this process
  (result))

(def-class event-rep-class (process-rep-class)
  (event-step)
  (meta-actions)
  (set-meta-action-rep))

;; a user action
(def-class action-rep-class (event-rep-class))

;; a web-site reaction
(def-class reaction-rep-class (event-rep-class))

;; these are performed by Woodstein and include setting variables, verifying data, etc.
(def-class meta-action-rep-class (event-rep-class))

;; namespace for variables is set as user and web-site actions are being recognized
(def-class var-rep-class (rep-class)
  (name)
  (expected-value)
  (value)
  (result-parent)
  (set-verb 'set)
  (verified))
```

Figure 4-5: The rep class.

actions, which we will see in the next section. It acquired data by retrieving pages for a reaction and analyzing them, as well as by analyzing the pages the user interacts with.

User Data in Process Explanations

In addition to representing user data, Woodstein also represents some data about the user's process. In the example, it sets the value for the retailer when the user loads the Amazin.com front page. It sets the shipper and credit card when each are selected by the user. This data is used when instantiating explanations. For instance, when creating an explanation for the process "user-placed-order-with-<retailer>" with the name string "You placed order with <Retailer>", Woodstein looks for any words surrounded by angle-brackets and looks them up as the symbols for variables. If they are bound to a value, such as "Amazin.com", then that value is used and the description becomes "You placed order with Amazin.com". If not, then the angle-brackets are removed and the description remains "You placed order with Retailer".

User Data Representation

Woodstein has a relatively simple representation for user and process data. Its knowledge-base is a global environment in which data items are added and retrieved by their symbols. Woodstein represents both primitive data, such as strings and quantities, as well as composite data which are associative-lists that have their own environments. For instance, a transaction amount is a number, while a transaction itself consists of a transaction amount, a description, the posting and transaction dates and so on.

How Woodstein Finds User Data in Pages

When Woodstein analyzes a page either that it has retrieved, or that the user loaded, it identifies any data in the page. If it doesn't have a value for the data item already, it extracts the value in the page. For instance, it learns what the name of the current item is when the user loads the item page for the book. Later, as it generates the process history tree, it knows whether this item name then becomes the value for a shopping cart item, and then an order item. If it has a value for the data item, as it does when it has an computed value for the order item name, it compares the new value in the page with its saved value. If they're different, it annotates the page data item as "maybe wrong". By the time Woodstein is done analyzing a page, it has a rep corresponding to each data item in the page.

4.6 How Woodstein Simulates and Verifies Web Site Reactions and User Data

We've seen how Woodstein uses the sequence of user actions to infer the overall process the user is engaged in. In this section we will see how, as it creates the process tree, it simulates the steps that web sites perform, then verifies both that they've been performed and that they produced the correct data. For example, when a user adds an item to the shopping cart, Woodstein sees in the process tree that the next step is for the web site to update the contents of the shopping cart. It simulates the update, and retrieves the shopping cart page to verify that the item has been added and the order subtotal has been correctly updated.

As we saw earlier, Woodstein learns that a web site reaction has occurred when it sees that the reaction updated a page. It knows that the web site recognizes the user's selection of shipper only when the selected shipper appears on a page, such as the order summary page. After parsing the user's actions to create an initial parse tree, Woodstein uses its process models to identify the expected web site reactions. It then simulates how the reactions process the user's data, and verifies the results when it can. It represents the processing performed by web sites with meta-actions that are described as part of the web site reaction step.

4.6.1 Step Meta-actions

There are two layers to Woodstein's representation of steps. Like other process reps, steps fit into the process hierarchy and feature a model process rep, a tense and a result. Beneath that layer is Woodstein's representation of how the step updates the user's data and how to determine whether the update was successful. User actions set data, when a user types text or clicks on a link, and web site reactions set data that must be verified, such as the updated contents of the shopping cart.

Some examples of meta-actions can be seen in Figure 4-4:

- "ws-set–shopping-cart-item" sets the "shopping-cart-item" to have the value of "current-item-name" and the source page at which this can be verified is "retailer-shopping-cart-page"

- "ws-set–total-price" meta-action sets the "total-price" to be the sum of "shopping-cart-subtotal" and "shipping-charge"
- "ws-set–order" creates the "order" composite data item with the "shopping-cart-item", "total-price", "shipping-address", "shipper", and "payment-method" data items.

4.6.2 Collecting Reactions and Simulating Meta-actions

After recognizing a new user action, Woodstein adds the action to the parse tree and reparses creating a parse tree featuring the user's action. Immediately afterward, it performs a depth-first traversal of the tree to identify subsequent web site reactions. It examines incomplete process nonterminals and collects the reactions that would occur before the next user action. It then simulates the meta-actions for these reactions, setting user data. When a meta-action has a verification page, Woodstein retrieves the page and checks whether the data on the page has the expected value. If so, it marks the reaction as complete and in the past tense. If not, it marks the process as still ongoing.

When it has found and simulated all web site reactions, Woodstein re-runs the parse with process parse tree and reactions as inputs to create a complete history of the process including both the user's action and web site reactions.

4.7 How Woodstein Manages Views of Process and Data History

Woodstein's interface features multiple views for visualizing different aspects of web processes.

Ordinarily, Woodstein works in the background and the only indication that it is running is the logo it adds to the bottom right of each page it is monitoring. When the user wants to inspect some data in the current page, clicking on the logo turns on the inspector. This reveals buttons for the reps in the page that Woodstein identified. Clicking on a rep causes the rep to become the "selected" rep.

Borrowing the convention from data-flow diagrams, process buttons are rounded while data buttons are rectangular.

4.7.1 The Saved Page View

The most important view Woodstein provides is its "saved page" view with pages loaded and seen by either the user or Woodstein itself. Every page the user interacts with is saved and added to the record of the user's action. Woodstein, guided by its process models, looks for a web site reaction and retrieves the page in which its result appears. For example, after the user adds an item to the shopping cart, it retrieves the shopping cart page featuring the updated contents.

Although other technologies are able to capture the pages of a user's action, they only allow searching through the sequence of pages and playing back the user's action. Only Woodstein present the pages within the hierarchy structured by the process itself. This allows a user to quickly "drill-down" to find a page he is looking for.

Unlike data and process reps, pages themselves aren't inspectable within Woodstein's interface. Instead, each data and process rep has an associated "source page" in which it appears:

Data item the saved page is the first page in which it appears

Abstract process the saved page is the help page in which it is described

User action the saved page is the page the user interacted with

Web-site reaction the saved page is the page in which the label for the set data item appears, usually with the data item

The source page for a rep is saved soon after it is created.

4.7.2 The "What's Happening" and "Why" Process History Views

We show the user's history as a hierarchy. We expect that computer users are comfortable with interacting with a tree and drilling down to see individual steps from their experience in using a file browser, and it uses similar icons to show the tree.

Selecting Buttons

Navigating the process history can be overwhelming, so we chose to present only as much information as is appropriate. Selecting a rep causes its subtree to open and its children be revealed. On the other hand, unlike a file browser tree, outside events cause a rep to be selected and de-selected frequently, as the user interacts with other views. Rather than leaving it up to the user to hide unnecessary subtrees, a selected rep's subtree closes automatically when it is deselected. Rep subtrees manually opened by the user, however, remain open when the rep is automatically deselected.

User Data as the Status or Result of a Process

The result or status of a process is presented next to it so that the user can easily judge whether it was successful and the data it produced was correct. A process' result is the last data set within it, such as the transaction that results from Amazin.com requesting a payment from Onlibank. If the process is incomplete and ongoing then the data item is its status. In placing the order, for instance, after adding an item to the shopping cart, the status of placing the order and of the entire purchase itself is the shopping cart subtotal computed by summing the total prices of the items in the cart.

Although it makes sense that the transaction is the result of the payment request, for adding an item it would seem to make more sense if the status of the order were the list of shopping cart items. However, because the subtotal is set after the shopping cart items are, it becomes the status. Similarly, if Amazin.com updated its website in response to receiving the payment, then that verification would become the result of the payment request process, and which may or may not be desirable. This is worth reconsidering, and another approach might involve having data items that may act as a status or result either specified or identified using heuristics.

Stepping through Processes

The process history views also supports navigating through the process history with buttons in the top frame of the window. The user can play the process history both forward and backward with these buttons. The user can also jump among his own steps with user-forward and user-backward buttons. Thus the user is able to see "what was the last thing I did before this happened?".

Each rep has a "step-parent" slot to keep track of its parent step, and a "steps" slot for its children steps. These are set as part of creating the process history.

The “What’s Happening” Process History View

Unlike the other views, the “what’s happening” view is accessed when user is browsing, outside of inspector mode. It is intended to give the user the “big picture” of the process he is performing and where this step fits in. In the introduction, we saw how users often find web sites confusing and it can be difficult when performing tasks to see what the next step should be. This view provides some help for remaining oriented, although more advanced solutions will be discussed in the chapter on future directions.

4.7.3 The “How” Data History View

The data history view, or How View, is a backward dynamic program slice presenting the history of the computation of a value. The root of the slice is the “centered” rep which is the first rep the user inspected. It is labelled in the diagram as the “Symptom”.

Rather than show the entire slice and all of the values used to compute the centered rep value, the tree is revealed only as the user clicks on visible leaves. Selecting a leaf causes the reps used in its computation to be revealed. This way, the user isn’t overwhelmed with a mass of buttons when trying to understand how some data was created.

Representing the History of Data

Data history relationships among data and processes are set when data is computed. Each data rep keeps track of the input data reps used to compute it in its “uses-children” slot. It also sets itself to be a user of the input data reps in their “uses-parents” slots.

Limitations of the Data History View and Areas for Future Development

We also considered showing the dynamic forward program slice for a data item. That would helpful for seeing the data that are later corrupted by an earlier incorrect rep.

The data history view only shows data items and web site reactions. It does not show a user’s actions in setting the data, nor does it enable the user to see the inputs and outputs of an abstract process. This could be reworked to allow the user to drill-down the data history view. For instance, he could see boxes for Amazin.com, Onlibank and Zeno’s Delivery with arrows showing the movement of data among them. The semantics and interaction structure for this is somewhat complicated, so we leave it be developed further in the future.

The graph for the data history view is rendered in HTML, using tables. Though it is represented as a graph, the actual output is a tree rooted at the symptom. This creates a problem when nodes appear multiple times, as when multiple data items in the Onlibank credit card transaction are computed from the Amazin.com order. Shared nodes like the order appear twice in the tree. A more robust implementation of the grapher would layout the graph and support incrementally revealing it as the user inspected farther back.

4.8 Guides the User in Diagnosing Errors

As we saw in the examples, Woodstein guides the user through the process of elimination in both top-down and bottom-up approaches to debugging. Once the user has identified a faulty process or data rep, Woodstein generates an email describing his process of diagnosis to send to the appropriate customer service representative. In this section we will see how it provides these forms of help.

4.8.1 Representations of Hypotheses

Woodstein keeps track of the user's judgements of which reps look correct and which look incorrect. In turn, it performs the process of elimination to suggest further reps that should be investigated. We call the set of these annotations a hypothesis.

Reps could be extended to support annotation by just adding another slot to the rep class. However, that approach raises some problems. The user might change his mind about a rep's correctness, so "undo" should be supported. Tracking the last annotated rep would work for one undo step, but not allow a sequence of annotations to be undone. Distinguishing between the user's annotations and Woodstein's complicates this even further. So the current hypothesis is kept as a separate record with the sequence of both the user's and Woodstein's annotations.

Instead of modifying rep objects directly, Woodstein maintains a linked list of "hypothesis" objects that refer to the rep they hold the annotation for. The current hypothesis is the front of the list and features the most recent annotation, whether by the user or Woodstein itself. The end of the list is the user's original annotation.

The first time the user uses a rep button menu to note that a data rep "looks wrong" or a process rep "looks unsuccessful", Woodstein creates a new current hypothesis. It also creates a current hypothesis when it fails to verify a web site reaction or when data in a page it analyzes differs from what it expects. It also opens the debugging trail view.

4.8.2 Debugging Trail View

The debugging trail view shows information about diagnosing a problem and the user and Woodstein's current hypothesis. Like other views, the debugging trail features a title bar with buttons. It has an "Undo" button allowing the user to undo the last annotation and a "Complain" button to create an email complaint, which we will discuss later. At the top of the view display is the diagnosis history with a summary of the current hypothesis at its list of hypothesis objects normally hidden below it. Below the history is the list of maybe-incorrect reps that Woodstein suggests the user examine to determine the correctness of. Finally, below that list is an area where Woodstein gives advice on how to determine whether a rep should be marked correct. Data item reps are correct if the value looks right. Process reps are correct if it looks like the problem started before they happened.

4.8.3 How Woodstein Identifies Maybe-Incorrect and Incorrect Reps

When the user annotates a rep, Woodstein marks the reps that caused it to be maybe-incorrect. An incorrect process has its children marked maybe-incorrect. A data item has the process that set it and the inputs to that process marked as maybe incorrect.

As the user examines and judges the correctness of maybe-incorrect reps, Woodstein's simple reasoning system performs the process of elimination to determine whether any maybe-incorrect reps should be marked as incorrect. If an incorrect process has all but one of its child marked correct, then the remaining child must be incorrect. If the generating process of an incorrect data item is correct, and all but one of its inputs is correct, then the remaining input must be correct. If all inputs to the process are correct but the result is incorrect, then the process must be incorrect.

4.8.4 How Woodstein Generates an Email Complaint

When the user has isolated the rep that caused the problem he is investigating, he can click on the debugging trail view's "Complain" button and Woodstein automatically generates a complaint email. In the email, Woodstein specifies that the last rep marked incorrect is the problem, and it outputs the current hypothesis as a sequence of incorrect and correct annotations. In this way, it provides some help to the user in resolving the problem.

Chapter 5

An Evaluation of Woodstein

We designed Woodstein with the intention of creating an interface that enables end-users to see the big picture of their actions on the web. As part of this, we developed visualizations of the user's actions and data, including a process history view. Though we focused on presenting the processes in a simple and easy-to-understand way, we can know whether we've succeeded only by having end-users interact with the agent.

In evaluating Woodstein, we sought to isolate and test its boldest claim. When consumers have problems, they are comfortable with, or at least acquiesce to, picking up the phone to figure out what went wrong. When they've had frustrating experiences with customer support in the past, consumers may try to diagnose problems themselves. We believe that the most controversial assumption embodied in Woodstein is that once end-users are familiar with how a new tool supports visualizing, understanding and diagnosing web processes, they will in fact be interested in using it to diagnose the problems they have on the web. We sought to discover this by testing a set of related issues.

5.1 Experiment Hypotheses

The most important issue we had to test was whether end-users can diagnose problems more successfully by working with the cooperation of a software agent than with the default option - trying to diagnose the problem for a few minutes before sending an email complaint. We hypothesized that *the group which used the agent for diagnosis would be more successful, with more participants succeeding and taking less time to succeed.* To test this, we created a scenario in which participants in both groups attempted to identify the most specific requirement not met for reaching some goal, in this case, for graduating. The test scenario was described as the graduation example.

Associated with this main comparison were some additional questions about the agent we wanted to answer. We wanted to know whether participants in the experimental group who used the agent were satisfied with it, and whether it seemed like it would be helpful not only for this problem, but for others they might have on the web. We asked the participants' opinions in post-test questionnaires.

5.2 Experimental Design

Each participant attempted to diagnose a problem involving information on web pages. Both groups used a version of the agent with some of the features deactivated. The control group diagnosed the problem using only information from the web site and used the agent only to send a complaint. The experimental group diagnosed the problem with the help of the process history view and also sent a complaint through the agent's interface.

The problem involved determining why the user, in the role of a student, was not eligible to graduate and the diagnosis required examining pages on a student account site to identify which requirement in particular has not been satisfied.

Upon arriving, participants received a general explanation of how the agent works and provided consent. They then answered questions about their use of the web and e-commerce, and their programming background.

Prior to each test scenario, participants received an explanation of their support option for the scenario. Participants in the control group were shown how to turn on the inspection mode and use the single menu option of noting that a data item "looked wrong". This generated an email in their email client that was sent to our test server.

Participants in the experimental group saw a live demonstration of the agent tracks the a user's action, with the first few steps of the order example visualized in the "what's happening" view. They then saw how the "why" process history view can be used understand the overall structure of the user's process, and revisit previously visited pages. They took a quiz in which they started from the user's credit card transaction page and found the item page for the purchased item, featuring both its image and price. During this quiz, the experimenter provided some help and answered questions about how to use the agent.

Participants in both groups then attempted to diagnose the problem. Participants in the control group were only able to select a rep button and complain, while participants in the experimental group could see the history of how the graduation eligibility was determined. Participants navigated the process history view and visited pages featuring relevant data including explanations of the requirements. The test concluded once a participant had either found the unsatisfied requirement, or time had run out.

Scripts and other documents used for the study can be seen in the appendices.

5.2.1 Study Segment Times

Control Group:

1. Introduction, Consent and Pre-Test Questions (5 minutes)
2. Explanation: Walkthrough in Diagnosing Purchase Scenario (5 minutes)
3. Test: Diagnosing Graduation Scenario (25 minutes)
4. Post-Test Questions (10 minutes)

Experimental Group:

1. Introduction, Consent and Pre-Test Questions (5 minutes)
2. Explanation: Walkthrough in Tracking and Diagnosing Purchase Scenario Location (20 minutes)
3. Quiz: Diagnosing Purchase Scenario Price (5 minutes)
4. Test: Diagnosing Graduation Scenario (15 minutes)
5. Post-Test Questions (10 minutes)

5.2.2 Role of Training for Experimental Group

In designing the experiment, we were concerned that the extra time spent training experimental group participants to use the tool might unintentionally also train them to diagnose problems in general more effectively. If this were the case, then any difference observed between the groups might be attributable to this extra training. First, we tried to control for that by using participants who have experience with web processes and who are likely to have already tried to diagnose and resolve problems they've had on the web. To further address this, we trained participants in both groups to use the agent for one domain with the purchase scenario and tested them in a completely different domain with the graduation scenario. Experimental group participants were told during training that within the view a process is shown beside its result, but during the test they were not told that the determination of whether the requirements were satisfied was a process consisting of sub steps and intermediate results.

5.2.3 Type and number of participants involved

Testing included 16 participants total, 8 in each group. Participants were members of the MIT community familiar with everyday e-commerce procedures such as online-banking, purchasing books at Amazon, etc.

5.2.4 Method of recruitment

Participants were recruited through advertising on campus.

5.2.5 Length of participant involvement

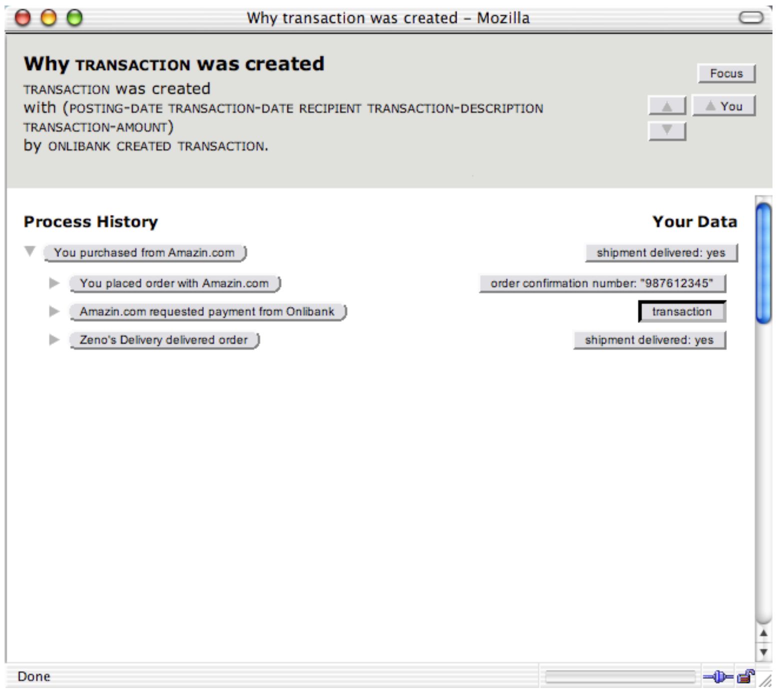
Participants participated for approximately 45 to 60 minutes.

5.2.6 Location of the research

The research was conducted in an office at the Media Lab. The office was equipped with a computer featuring the Woodstein agent software and a web browser. We also used software for recording the screen and audio inputs, and email software for sending complaints to out test server.

5.2.7 Procedures for obtaining informed consent

Participants were asked to sign a paper consent form and click-through the same form on a web-page. This form indicated that they were aware of the circumstances of the study, including the audio recording of conversations with the experimenter. It is included in the appendix.


Why transaction was created – Mozilla

Why TRANSACTION was created

TRANSACTION was created
with (POSTING-DATE TRANSACTION-DATE RECIPIENT TRANSACTION-DESCRIPTION TRANSACTION-AMOUNT)
by ONLIBANK CREATED TRANSACTION.

Process History | **Your Data**

- You purchased from Amazin.com — shipment delivered: yes
  - You placed order with Amazin.com — order confirmation number: "987612345"
  - Amazin.com requested payment from Onlibank — transaction
  - Zeno's Delivery delivered order — shipment delivered: yes

Figure 5-1: Initial state of process history view in quiz

5.2.8 Procedures to ensure confidentiality

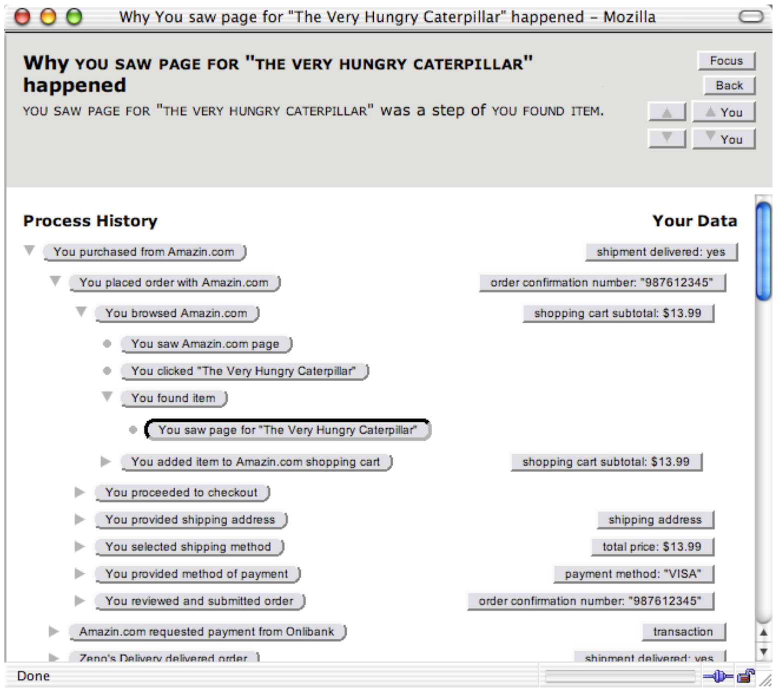
Participants were asked general demographic data, and about their level of familiarity with using the web, e-commerce and programming. Participants were also asked about e-commerce related problems they may have had. No other personal information was collected.

5.2.9 Quiz Details

Upon initially inspecting the transaction, the process history view appears as in Figure 5-1. The quiz involves drilling-down through placing the order, browsing and finding the item, as shown in Figure 5-2.

5.2.10 Test Details

Upon initially inspecting the graduation status, the process history view appears as in Figure 5-3. The test involves drilling-down through the Institute's computation of the student's satisfaction of requirements, as shown in Figure 5-4.



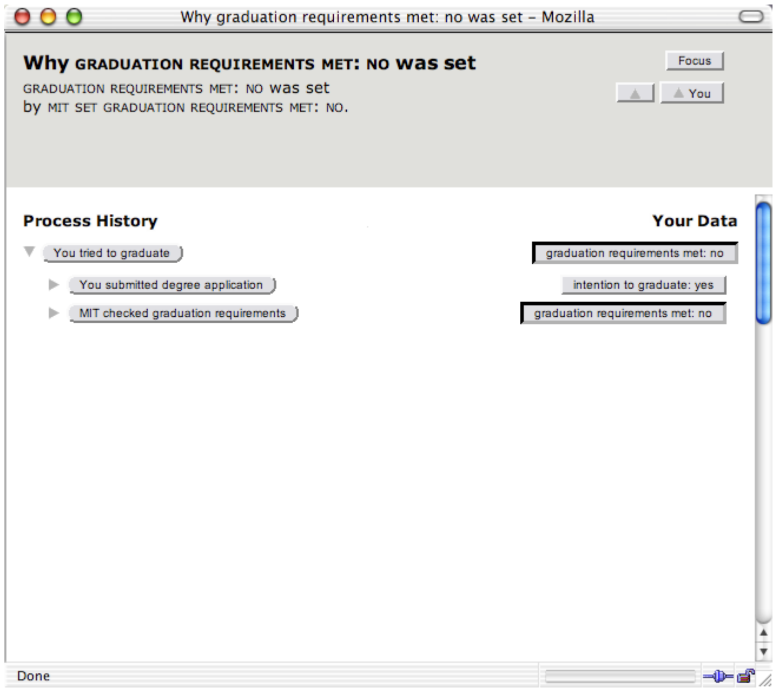
Why You saw page for "The Very Hungry Caterpillar" happened – Mozilla

Why YOU SAW PAGE FOR "THE VERY HUNGRY CATERPILLAR" happened

YOU SAW PAGE FOR "THE VERY HUNGRY CATERPILLAR" was a step of YOU FOUND ITEM.

You purchased from Amazin.com — shipment delivered: yes
You placed order with Amazin.com — order confirmation number: "987612345"
You browsed Amazin.com — shopping cart subtotal: $13.99
You saw Amazin.com page
You clicked "The Very Hungry Caterpillar"
You found item
You saw page for "The Very Hungry Caterpillar"
You added item to Amazin.com shopping cart — shopping cart subtotal: $13.99
You proceeded to checkout
You provided shipping address — shipping address
You selected shipping method — total price: $13.99
You provided method of payment — payment method: "VISA"
You reviewed and submitted order — order confirmation number: "987612345"
Amazin.com requested payment from Onlibank — transaction
Zeno's Delivery delivered order — shipment delivered: yes

Figure 5-2: Final state of process history view in quiz



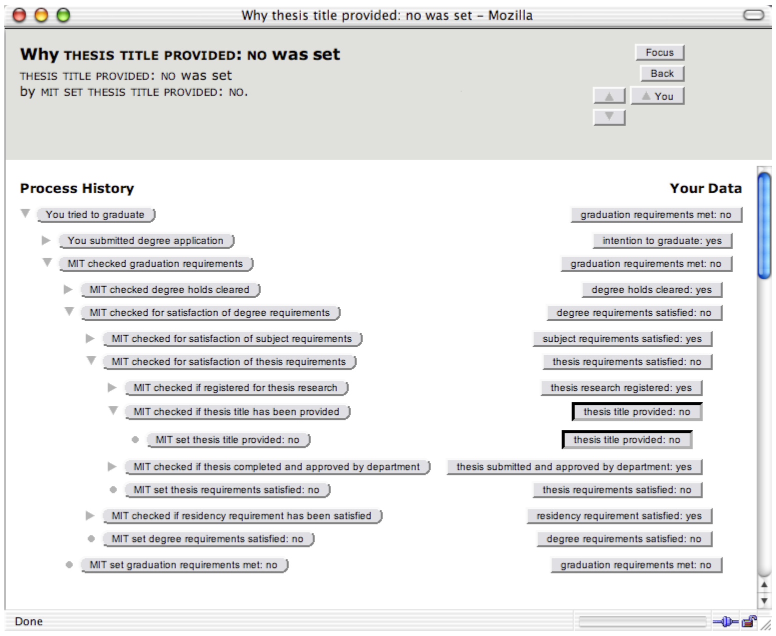
Why graduation requirements met: no was set – Mozilla

Why GRADUATION REQUIREMENTS MET: NO was set

GRADUATION REQUIREMENTS MET: NO was set by MIT SET GRADUATION REQUIREMENTS MET: NO.

You tried to graduate — graduation requirements met: no
You submitted degree application — intention to graduate: yes
MIT checked graduation requirements — graduation requirements met: no

Figure 5-3: Initial state of process history view in test


Why thesis title provided: no was set – Mozilla

Why THESIS TITLE PROVIDED: NO was set

THESIS TITLE PROVIDED: NO was set
by MIT SET THESIS TITLE PROVIDED: NO.

Process History	Your Data
You tried to graduate	graduation requirements met: no
You submitted degree application	intention to graduate: yes
MIT checked graduation requirements	graduation requirements met: no
MIT checked degree holds cleared	degree holds cleared: yes
MIT checked for satisfaction of degree requirements	degree requirements satisfied: no
MIT checked for satisfaction of subject requirements	subject requirements satisfied: yes
MIT checked for satisfaction of thesis requirements	thesis requirements satisfied: no
MIT checked if registered for thesis research	thesis research registered: yes
MIT checked if thesis title has been provided	thesis title provided: no
MIT set thesis title provided: no	thesis title provided: no
MIT checked if thesis completed and approved by department	thesis submitted and approved by department: yes
MIT set thesis requirements satisfied: no	thesis requirements satisfied: no
MIT checked if residency requirement has been satisfied	residency requirement satisfied: yes
MIT set degree requirements satisfied: no	degree requirements satisfied: no
MIT set graduation requirements met: no	graduation requirements met: no

Figure 5-4: Final state of process history view in test

5.3 Study Results

In this section we present the results.

5.3.1 Questionnaires

Participants filled out the same questionnaires both before and after interacting with the agent.

	Control	Experimental
Male	3	6
Female	5	2
no answer	0	0

Table 5.1: Gender

	Control	Experimental
less than 20	4	1
20 to 30	4	7
30 to 40	0	0
40 to 50	0	0
50 to 60	0	0
over 60	0	0
no answer	0	0

Table 5.2: Age Range

	Code	Control	Experimental
less than 1	1	0	0
1 to 5	2	1	0
5 to 10	3	2	1
10 to 20	4	1	2
20 to 30	5	3	2
40 and up	6	1	3
no answer		0	0
median		4.5	5
range		4	3

Table 5.3: "How much do you use the web, in average hours per week? If you use the computer a lot and are usually online, how much do you use the computer per week?"

	Control	Experimental
web email (Hotmail, Yahoo! Mail)	8	8
online shopping (Amazon.com, Buy.com)	8	8
online auctions (Ebay.com)	4	5
online banking	6	4
no answer	0	0

Table 5.4: "Which of these actions have you *ever* done on the web?"

	Control	Experimental
web email (Hotmail, Yahoo! Mail)	8	8
online shopping (Amazon.com, Buy.com)	4	7
online auctions (Ebay.com)	1	2
online banking	6	4
no answer	0	0

Table 5.5: "Which of these actions do you do at least *once per month* on the web?"

	Control	Experimental
web email (Hotmail, Yahoo! Mail)	8	7
online shopping (Amazon.com, Buy.com)	3	2
online auctions (Ebay.com)	0	0
online banking	4	2
no answer	0	0

Table 5.6: "Which of these actions do you do at least *once per week* on the web?"

	Code	Control	Experimental
0	1	0	0
1	2	2	2
2 to 3	3	2	4
more than 3	4	4	2
no answer		0	0
median		2.5	3
range		2	2

Table 5.7: "Have you taken any classes involving programming? If so, how many?"

	Code	Control	Experimental
never	1	1	0
less than 1	2	1	1
1	3	3	2
2 or more	4	2	3
no answer		0	0
median		3	4
range		3	2

Table 5.8: "How many years have you programmed?"

Doing Actions on the Web
We'd like to know about people's experiences with doing actions on the web such as placing orders, making payments, signing up for classes, and so on.
We've all had the experience of doing an action on the web and thinking that it went well, but later finding out it didn't. Maybe you ordered something online and got something else, or you approved some transaction that didn't happen.
Tell me about a time when you had to interact with some bureaucracy like a corporation or government to do an action on the web. Originally, you thought it had worked, but it didn't, and you didn't find out until later.

- First, what was supposed to happen?
- What went wrong? How did you find out about it? How did you fix it?
- What kind of help would you have liked, in recognizing that something went wrong, or even in fixing it?

Table 5.9: Other questions asked before the test

5.3.2 Test Results

	Control	Experimental
success	2	8
failure	6	0

Table 5.10: Success in selecting the correct requirement

	Experimental								Control	
	3:05	3:20	3:50	4:00	5:15	6:40	6:40	7:15	14:30	18:00
mean	5:00								16:15	
standard deviation	1:40								1:45	

Table 5.11: Times to select the correct requirement (in minutes and seconds)

All participants in the experimental group were successful in completing the task (Table 5.11). Only two in the control group were successful. The participants in the experimental group took about 5 minutes, while the two successful control group participants took an average of over 16 minutes. The slowest participant in the experimental group took half as long as the fastest participant in the control group.

Because of the small sample sizes, particularly the small number of control group participants who were successful, and the absence of experimental group members who did not succeed, we are unable to determine the statistical significance of this result. Thus we interpret the results as suggestive of a difference but not proof of one.

5.3.3 After-Test Questions

	Code	Control	Experimental
strongly agree	7	0	1
agree	6	0	3
somewhat agree	5	0	1
neutral	4	0	3
somewhat disagree	3	1	0
disagree	2	3	0
strongly disagree	1	4	0
no answer		0	0
median		1.5	5.5
range		2	3

Table 5.12: “It was easy to diagnose the graduation problem”

	Code	Control	Experimental
strongly agree	7	0	2
agree	6	0	4
somewhat agree	5	0	0
neutral	4	0	1
somewhat disagree	3	1	1
disagree	2	4	0
strongly disagree	1	3	0
no answer		0	0
median		2	6
range		2	4

Table 5.13: “It took the right number of steps to diagnose the graduation problem”

In responses to later questions, participants in the experimental group suggest that the agent provide relevant information more easily (Table 5.13). However, here we see that most agreed that the task took the “right number” of steps.

	Code	Control	Experimental
strongly agree	7	0	1
agree	6	0	4
somewhat agree	5	0	2
neutral	4	2	0
somewhat disagree	3	1	1
disagree	2	1	0
strongly disagree	1	4	0
no answer		0	0
median		1.5	6
range		3	4

Table 5.14: “The steps I had to take for diagnosing the graduation problem were intuitive”

Most of the participants in the experimental group agreed that the steps for the task were intuitive (Table 5.14).

Most of the participants in the experimental group agreed that they felt they understood how to use the agent effectively (Table 5.15).

Notably, a majority of the participants in the experimental group agreed that they'd like to have an agent like this for credit card problems, with half strongly agreeing (Table 5.16). One participant in each group expressed reservations in response to a later questions. The participant in the control group felt “uncertainty that the issue will be resolved, because you don't have instant feedback like you do when talking to a person”. The participant in the experimental group was concerned about the privacy of one's credit card information

	Code	Control	Experimental
strongly agree	7	1	1
agree	6	3	4
somewhat agree	5	1	2
neutral	4	1	0
somewhat disagree	3	2	1
disagree	2	0	0
strongly disagree	1	0	0
no answer		0	0
median		5.5	6.0
range		4	5

Table 5.15: "I feel like I understand how to use the agent effectively"

	Code	Control	Experimental
strongly agree	7	0	4
agree	6	4	2
somewhat agree	5	1	1
neutral	4	2	0
somewhat disagree	3	0	0
disagree	2	0	1
strongly disagree	1	1	0
no answer		0	0
median		5.5	6.5
range		5	5

Table 5.16: "I'd like to have an agent like this one to diagnose and resolve problems involving my credit cards"

and thought that this might be too much sensitive information to have on one's computer - "I suspect that storing these kinds of things on the computer will later give strangers an opportunity to use them".

	Code	Control	Experimental
strongly agree	7	0	3
agree	6	3	2
somewhat agree	5	2	2
neutral	4	1	0
somewhat disagree	3	0	1
disagree	2	1	0
strongly disagree	1	1	0
no answer		0	0
median		5	6
range		5	4

Table 5.17: "I'd like to have an agent like this one to diagnose and resolve problems like the graduation problem"

	Code	Control	Experimental
between 0 and 5 minutes	1	1	0
between 5 and 15 minutes	2	2	3
between 15 and 30 minutes	3	3	2
between 30 and 60 minutes	4	2	2
between 1 hour and 2 hours	5	0	0
more than 2 hours	6	0	1
median		3	3
range		3	4

Table 5.18: "Based on your experience with similar problems, how long do you think it would have taken to diagnose and resolve the graduation problem over the phone? (Including time spent on hold)"

With the question for Table 5.19, we wanted to discover how short the resolution with phone support would have to be before the participant used the agent. Half of the participants in the experimental group would have first turned to the agent if the phone call took more than 5 minutes, all but one if more than 15. The remaining participant in the experimental group noted that he/she did not anticipate using a agent like this one for support in the future.

With the question for Table 5.21, we wanted to discover how short the resolution by email would have to be before the participant used the agent. Most of the participants in the experimental group would have first turned to the agent if they had to send more than

	Code	Control	Experimental
between 0 and 5 minutes	1	2	4
between 5 and 15 minutes	2	3	3
between 15 and 30 minutes	3	0	0
between 30 and 60 minutes	4	3	0
between 1 hour and 2 hours	5	0	1
more than 2 hours	6	0	0
median		2	1.5
range		3	4

Table 5.19: "Suppose you knew in advance how long it would take to diagnose and resolve the graduation problem on the phone. How long would it have be before you would try to use an agent like this one first? (Assume that the graduation problem was resolved after sending the first email with the system)"

	Code	Control	Experimental
1 email	1	0	0
2 emails	2	2	3
between 3 and 4 emails	3	4	5
more than 5 emails	4	2	0
median		3	3
range		2	1

Table 5.20: "Based on your experience with similar problems, how many emails do you think you would have had to send to diagnose and resolve the graduation problem by email? (This includes emailing one person, who tells you to email someone else, and so on.)"

	Code	Control	Experimental
1 email	1	4	2
2 emails	2	2	4
between 3 and 4 emails	3	2	2
more than 5 emails	4	0	0
median		1.5	2
range		2	3

Table 5.21: "Suppose you knew in advance how many emails it would take to diagnose and resolve the graduation problem. How many would it have be before you would try to use an agent like this one first? (Assume that the graduation problem was resolved after sending the first email with the system)"

2 emails.

- What was the most challenging part of the problem?
- When did you feel stuck and what did you do about it?
- How do you see this as different from the normal ways of trying to diagnose similar problems on the Web?
- What did you like about the help the agent gave you?
- What did you dislike about the help the agent gave you?
- What would you like the agent to do differently?

Table 5.22: Other questions asked after the test

The questions in Table 5.22 elicited many positive comments about the help provided by Woodstein, as well as some concerns. Participants liked the overall approach in tracking actions, saving pages the user interacted with and inspecting data directly:

- "The regular data turned into buttons was great."
- "I liked that it saved the pages that I viewed. Often, the secure forms for purchasing an item get lost, or you forget what you purchased, or how much they cost. This was a good way to organize your actions and use them as a reference for later. The ability of the agent to turn regular data into buttons and actions was really great. It saved a lot of time which would have been spent searching for the correct pages with the correct information."

Some liked the hierarchically structured history and showing the results associated with the action:

- "The agent not only remembered the entire history of the transaction, but also helped me browse through them easily. It did so by providing me with an expandable hierarchy of options (instead of simply flooding me with a list of URLs in chronological order)."
- "I like that all the steps where listed and I could see very easy where the problem was."

Some compared it positively to email and phone support:

- “It allowed me to pinpoint the problem in one sitting, without having to email or call people and wait for their responses.”
- “It allowed for ’instant gratification’; in solving the problems. I did not have to talk to anyone or wait for a response from them.”

Participants also mentioned some shortcomings. Their main concern was that there still may be too much information, and the agent doesn’t provide enough help in sorting through it:

- “too many things to click, with redundant information ←; too many choices to make!”
- “the mechanism for working the agent at times [presented] a bit too much information”

Participants also made specific suggestions for improvements to the agent. We will see those discussed with other future directions in the next chapter.

5.4 User Study Summary

In the comparison of the two groups, our hypothesis stated above was confirmed. Interestingly, half of the participants in the experimental group strongly felt they’d like to have an agent like this one for their credit card transactions, and half would have used the agent to diagnose a problem if it were to take longer than 5 minutes on the phone.

One concern about this data, however, is that all of the participants in the experimental group had some programming experience; most had programmed for at least one year. This suggests that the sample we selected, members of the MIT community, is not likely to reflect the population of web-users in general. Although we intended to develop an interface for end-users to understand processes they’re involved in on the web, future testing with subjects having less programming experience will test this more specifically.

Chapter 6

Conclusion and Future Directions

Woodstein was developed as an agent to help end-users understand their processes online. It was influenced by work in software agents, plan recognition, debugging, and other areas. It represents and visualizes a user's data and actions, as well as the reactions of web sites in providing an integrated view of web processes even when they span multiple web sites. In evaluating it, we found that study participants who used Woodstein were more effective in diagnosing a problem on the web than those who didn't. Half of the participants who used the process history view strongly agreed that they'd like to have an agent like this one for monitoring their credit card transactions, and half also would use it if they expected diagnosing a problem over the phone to take more than 5 minutes. However, with nearly all of the participants having some programming experience, more testing should be performed to see whether similar results are found with non-programmers.

Our early testing of Woodstein has yielded promising results and also pointed the direction to new improvements in the interface and implementation of the agent. In the next section we will examine the suggestions of the user study more specifically. Later in the chapter, we will look at possible improvements in other areas and for other problems.

6.1 Improvements Suggested by the User Study

The user study generated many ideas for improving Woodstein's interface. Although the reactions of the study participants in the experimental group were positive overall, they had many suggestions. Further, as the experimenter watched participants using the agent, he recognized alternate ways the agent's interface could behave and talking with participants also yielded suggestions.

We divide the improvements into two categories. There are fixes that will be made to the appearance of the interface. There were also more substantial changes to its implementation that may guide future development of the agent, and at least serve as problem areas that need to be clarified. The discussion for each improvement begins with quotes by the participants themselves, when the there was discussion of the issue.

6.1.1 Minor Changes Suggested by the User Study

The fixes included the following suggestions:

- Clarify the wording, particularly associated with accessing the views

  "The menus obtained by pressing down on the buttons were mostly the same, and the wording of the options made them not very easy to understand. So I found myself mostly clicking on the buttons, or, if that did not elicit the desired response, choosing the first option of the menu."

  Some of the participants didn't think "why" was a particularly good fit with the information shown in the process history view, while others felt the phrasing overall could be improved. We originally didn't want to burden users with having to figure out how to match their questions to the particular tool for answering it. Instead, we want them to just select a question and be able to find the information they're looking for. However, if the questions themselves don't clearly match the tool, then we may want to either add an indication of the view to the question, such as "Why was this set? (Process History)" and "How was this set? (Data History)", or just get remove the question entirely and leave only the view description.

- Visually distinguish saved pages the user visited from pages Woodstein retrieved

  "[add] something to separate the help page from the actually seen or needed pages"

  Clicking on a rep button in the interface causes the relevant saved page to be shown. However, the button itself doesn't indicate how Woodstein obtained the page. It may be a page the user actually interacted with, a page Woodstein retrieved for a web site reaction, or it may be a page for an abstract process that the agent retrieved for its description of the process. The way the page was saved could be indicated by the appearance of the button. For instance, at the left end of a button, before its text, there could be an icon indicating which type of page the saved page for this rep is.

- Associate colors with different data values, for instance have “no” a different color from “yes”

  “The only dislike was the visual aspect of the history. I would have liked there to be some sort of color-coding so that I can distinguish certain levels of actions from others. The grey buttons seemed to just blend together after a while... I think the buttons for the history should be color-coded, so it’s easier on the eyes.”

  Participants used the agent without the intelligent debugging support enabled so all of the buttons were the same color of grey. Some noticed the uniformity and suggested that the data buttons be different colors depending upon the value of the data. For instance, “no” could be a different color from “yes”. This would have been helpful for the graduation scenario, in which the participant could instantly see the unsatisfied requirement. More generally, it would be helpful for other rule-based processes, in which the set of values are limited and each could have a distinct color. This would be limited to discretely valued data, however, and wouldn’t scale for data in general. For instance, what color should an address be? Processes would also remain grey unless they inherited the color of their status or result.

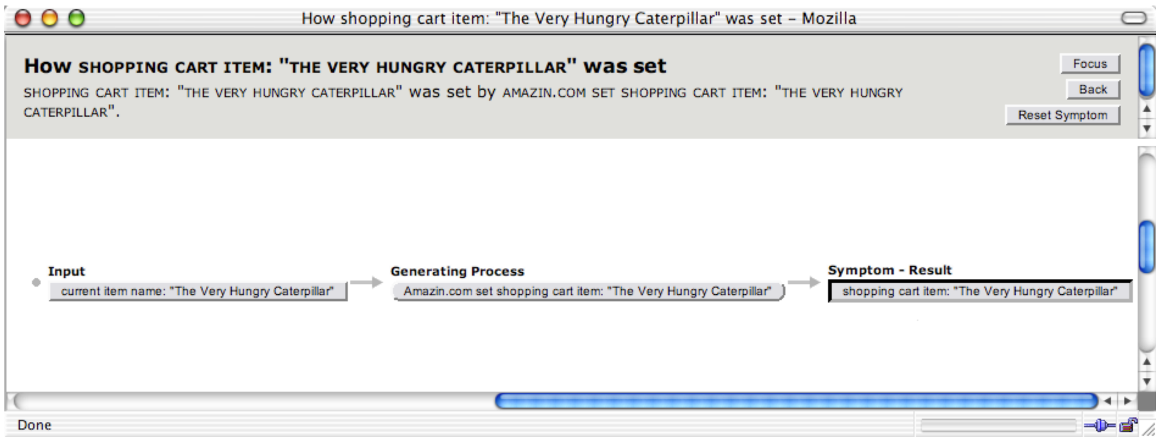
6.1.2 Major Changes Suggested by the User Study

More substantial changes were suggested either by the participants themselves, or their actions:

- Click on data to go to its source

  In the purchase scenario, the order confirmation features the book name. As the data history view in Figure 6-1 shows, the “shopping-cart-item” data item received its value from the “current-item-name” data item back when the user saw the page for the book and clicked to add it to the shopping cart. That these two data items have the same value created some confusion among participants, suggesting a different way of navigating through the data history.

  In the study, participants in the experimental group saw how to return from the


How shopping cart item: "The Very Hungry Caterpillar" was set – Mozilla

How SHOPPING CART ITEM: "THE VERY HUNGRY CATERPILLAR" was set

SHOPPING CART ITEM: "THE VERY HUNGRY CATERPILLAR" was set by AMAZIN.COM SET SHOPPING CART ITEM: "THE VERY HUNGRY CATERPILLAR".

Input: current item name: "The Very Hungry Caterpillar" → Generating Process: Amazin.com set shopping cart item: "The Very Hungry Caterpillar" → Symptom - Result: shopping cart item: "The Very Hungry Caterpillar"

Figure 6-1: History of "shopping-cart-item" data item.

credit card transaction page (Figure 2-1) to the order confirmation page (Figure 2-4) in the purchase scenario. Then for the quiz, they were started at the transaction page and asked to find the original page for the item ordered. Participants typically returned to the order confirmation and several were observed clicking on the name of the item, expecting to be taken back to the page for the item. The order confirmation page features the "shopping-cart-item", however, not the "current-item-name" that appears on the book page.

Clicking on a rep's button in the agent's interface loads the saved page for the rep. If a user clicks on a data item in a page, and the data had appeared before on an earlier page, the earlier page will be loaded. In the case of the purchase scenario, however, though the value is the same, the data item with the value differs. The behavior of the participants suggested that this could be made more clear. It does seem intuitive, however, that clicking on a value in pages should load the previous page in which it appeared. When a data item just copies the value of another, there should be an easy way to jump to the previous data item without having to open the data history view. However, it's unclear how to do this when some value is computed from multiple values, such as with an order total that is the sum of the prices and shipping charge. This should be explored further.

- Allow browsing during inspection mode

  "Once I got to a site close to where I wanted to be, I couldn't just [click] to the next one since the links didn't work. As such, I had to return to the agent each time I slightly missed in order to find the correct page. It ought to have cached pages that allow links rather than what is essentially a webpage that has become a word file."

"not allowing the use of links within the saved pages made for needless returning to the agent."

The current version of Woodstein distinguishes between browsing and inspection mode (although the "what's happening" view updates and allows saved pages to be viewed during browsing). It was recognized that this distinction is somewhat messy and should be either be further refined for clarity or removed entirely. During inspection mode, the history of the process can be played back with buttons in the title bar of the process history view. However, multiple users made the suggestion that web pages being inspected should be live, an interesting possibility that we didn't consider. A future version of Woodstein should take this into account and perhaps merge the two modes, allowing users to traverse the history either by using the agent's rep buttons, or clicking on the links to explore alternate paths.

- Include a help system

  "There is no help for the agent, so I had to spend time to learn how to use it.".

  Somewhat ironically, though the agent is intended to help the user understand processes on web pages, it doesn't itself have a help system. It currently only explains what will be the immediate result of a user clicking or selecting a menu item in the status bar during inspection mode. A traditional help system for using the agent could be incorporated, but there's also the possibility for more dynamic help, just as the agent provides dynamic help for the processes and data it tracks. In particular, the agent could perform plan recognition on the user's interaction with it and provide help for inspection as well as browsing.

- Provide alternate ways of interacting with user data

  "use 'to', 'transaction date' etc. to sort items in list (like in email) rather than be redundant to the information in each smaller button underneath."

  Some participants felt the agent should simplify how processes and data are presented, or allow even more sophisticated means for interacting with them. They suggested allowing the user to search through the history either to narrow down the problem, or just for review old actions. For instance, a user could see that something is wrong with some data and say “I think the problem may be in something related to X”. The agent has the “how” data history view for viewing how data was set, but it wasn’t turned on for the testing.

- Automatically diagnose problem sources

  “maybe there could be a program which highlights certain trouble-markers, such as any action which received a ’no’ rating.”

  The agent itself could play a more active role in identifying problems. We will have more to say about this in a later section.

- Simplify how an action of setting data is associated with a goal

  Some participants found it redundant that a process is described, such as “MIT checks graduation requirements”, and then has the last step of setting the data, “MIT sets graduation requirements”. One mentioned that it was strange to look at the top of the subtree to see the description for the process and the bottom to see the last step that set the data result. The top process shows its result data item on the right in the process history view, however. Further, the hierarchy of processes itself matches when they happened temporally - “MIT sets graduation requirements” must happen after all intermediate data has been computed. However, this approach does seem somewhat redundant, so a simpler presentation of how abstract processes will be considered.

6.2 Help Users in Managing Business Relationships

Woodstein could be further developed to make the user’s transaction data useful in other ways, besides directly explaining to the user how a transaction has proceeded. We think of these as tools for managing relationships with businesses, by analogy with Customer Relationship Management (CRM) tools. They could be viewed as a component of the debugging system, but they are also useful in their own right.

This data could be put to use whenever the user is required to provide information about a transaction. For instance, if a user had to fill out a form on a web page requesting help or more information, the agent could automatically identify and retrieve the relevant data and enter it as the default form values for the user. Not only would this ensure that the data is correct and consistent with the rest of the transaction information, the user would be spared the inconvenience of having to look up this information and enter it manually. Simple facilities in Web browsers currently help you fill out forms by making available the last entries you typed, or matching to a pre-stored form, but tracking transactions would provide more context-sensitive prompts and completions.

As part of tracking information the user enters, Woodstein could track who knows what. We've all experienced the hassle of having to update contact information when we've moved, switched jobs, or experienced other changes. An agent that tracked what information we've entered where would know exactly who needs to be updated. It would be even more helpful if it could perform the updates automatically.

More generally, Woodstein could help a user by automatically performing common actions. A user could ask it to schedule a trip according to a set of parameters, and, with its models of the web actions involved, it could fill out the data in web pages to buy a plane ticket, reserve a hotel room and rent a car, while saving records of what it did, perhaps to the user's employer for reimbursement.

6.3 Manage Contextual Information about a User's Actions

A user performs actions with intentions at an even higher level than what Woodstein represents. For example, a user might perform an action of planning a trip including buying a plane ticket, reserving a hotel room, and renting a car. With knowledge about these actions, the agent could also recognize that the user is scheduling a trip which sets up a context for the charges that happen during the duration trip. Eventually, it could support the user asking about the context of a charge, "What is this charge about?", and explaining "That was from you trip to Florida".

6.4 Act as a Repository for the Status of a User's Debugging

Diagnosing and resolving a problem may extend over time. Though the user sent a complaint about the incorrect graduation requirement, for instance, he has to wait until he hears back from the institution before being sure the problem is solved. Thus it would be helpful if Woodstein were able to support this as well, by keeping track of the associated emails, for instance.

6.5 Support User Training of Process Models

Woodstein requires process models and page descriptions to recognize the actions a user performs and the pages he visits. Currently, Woodstein accepts process models customized for a particular web site, but it may be possible to acquire them from the user himself, perhaps through training the agent in a programming-by-example style[30]. For instance, the user could perform a web action, highlighting and selecting data when it appears and explaining the agent how it relates to the process and how it is computed. For example, the user could select some text in the browser window and tell the agent, "this is the subtotal label, this is the subtotal data", etc. Or maybe the user would put it into recording mode, then explain the structure of the plan just performed. He could explain the sequence of his own actions, and the expected web site reactions. In this way, the agent would learn the process models through the user's intervention.

6.6 Support Multiple Simultaneous Actions

Multiple, interleaved user actions could require guidance from the user for recognition. For example, a user might be just about to confirm the purchase of an item, then decide to visit additional sites to reassure himself about purchasing that particular item. This would require the user telling the agent "I still want to see more opinions about this" to let it know that the user knows the purchase is incomplete and will be returned to.

Though Woodstein doesn't have the feature, it would be possible to support a user performing multiple, distinct actions, as long as each would be associated with a different browser window. It would keep track of each page separately and assign the actions performed at the page to the appropriate stream for recognition.

6.7 Help User in Reasoning about Hypothetical Possibilities

One of our original design goals was to support users in thinking about hypothetical possibilities involving their actions on the web. Woodstein is able to explain why actions happen, but users should be able to see why something did not happen, as well. Part of the first example we saw was based on that idea: "why hasn't my order been delivered yet?", but in order to discover the reason, the user had to look at the action and compare it with his expectations for what was happening. An important direction for this work would be to automatically generate explanations for why something did not happen, or why it happened differently from what was expected. I might ask why an item didn't appear in order and find out that it was not included because though I visited the item's page, I forgot to add it. An agent would compare what did happen with what would have had to have happened to generate the hypothetical case, then tell the user where the cases differ and at what point they diverged

The agent could also help the user understand the future consequences of decisions made now. In the example of placing an order, the agent updates the value for the shipper as soon as the user clicks the "ship" button to confirm. Alternately, it could display the different outcomes resulting from selecting different shippers. In the process view, the description could appear "Zeno's Delivery will deliver item on December 26", for instance. As the user continues and refines the order, the agent's expectations for the future could become more specific, enabling the user to see exactly the outcome to expect. This could be invaluable when performing actions across multiple sites, as in the case above of planning for a trip.

6.8 Integrate with the Semantic Web and Web Services

Though it is now capable of supporting minimally annotated HTML, an important priority is for Woodstein to evolve to support the possibility of richer annotations offered by XML and the Semantic Web. Indeed, it offers a vision of a user-centered interface to data on the Semantic Web. XML, by allowing the data within pages to be given semantic annotations, will undoubtedly benefit users by allowing higher-level interaction and querying. We see our approach as complementary to this vision in providing users help in managing the data they create and interact with as they perform actions on the Web. Though it will be nice to know that a particular piece of text within a transaction description is a dollar amount or date, how will the user be able to interact with this data in a meaningful way? Woodstein's approach is unique in allowing the user to see the history of the larger process that that data is a part of and, for example, to jump back to causally related pages where it has previously appeared or even the page that originally set it.

There is a growing realization of the need for business processes to be standardized and described in a uniform manner. However, it's not enough that business processes be standard and accessible in theory, it's also important that they be accessible in practice, preferably in a just-in-time manner. Woodstein matches user actions with abstract business process descriptions and is able to show the user the process their actions are a part of, as well as the overall structure of this process. Research in software debugging has demonstrated that showing the history of a particular execution of a process, complete with all of the details and data, is an effective way of explaining its dynamic behavior. Further, we expect that this approach will be useful for explaining the abstract processes themselves in light of a particular example.

Though this approach may not benefit consumers in the near term, we expect it to be immediately relevant within organizations. The ability for an individual within an organization to see that some data on an internal Web page is wrong, then load the inspector to see the structure of the individual's action, as well as the particular details of the processes triggered by the action, even if they occur elsewhere in the organization, will be invaluable.

6.9 Cooperate with Customer Support

Sometimes, it won't be possible to completely solve a problem from the user's perspective alone, because the answer might depend on details of the process or data that are internal to the vendor and are inaccessible by the user. In this case, he must resort to communicating with customer support, via e-mail or phone. Even in this case, Woodstein helps the user by summarizing his problem diagnosis process for communication by email.

We envision that a similar agent could be available to customer support representatives (csr), so that that csr can employ a similar debugging strategy. The user's agent could communicate automatically to the support person's agent, which could instantly access the details of the problem, and see the investigation the user has done thus far. Valuable phone support time would not be wasted reciting account numbers, checking and rechecking the same sequence of possible causes, and retelling the story to different personnel when the problem is escalated to a manager or more support people otherwise become involved. The support person's interface could be far more detailed and customized to the business than the general public would be willing to tolerate. Typically, the user does not want to need to know the details of the company's internal process, but such knowledge might be essential to solving the problem. Like the end-user, the csr could benefit by breaking down the problem into steps, verifying each independently, tracking the history of individual data items, and so on - all functions that Woodstein provides.

6.10 Teach about Diagnosing Problems in General

Woodstein guides the user in isolating an unsuccessful process or incorrect data through the process of elimination. It explains how to judge the correctness of reps in the "Debugging Trail" view, but doesn't explain how to do diagnosis and debugging in general. General strategies for diagnosing problems could be accessible through a help system.

6.11 Maintain and Share Histories of Problematic Data and Processes

Woodstein could also provide the user more help regarding individual processes and data. It assumes that process models include email addresses to complain to about each rep, but there is a possibility for process models to specify some information for each rep about how to diagnose any possible problems involving it, or its history of problems in the past. This information could even be shared among users, so that if many users discover that the default shipping option for purchases at Amazin.com may cause problems later. When the user first asks why his item hasn't arrived, the agent could suggest possible problems including this one. This information could even be fed back into the agent's functionality for explaining the results of a user's choices. Before confirming the shipment method, the agent could warn the user that other users have had problems when accepting the default shipper. This peer-to-peer annotation of processes would be parallel to the peer-to-peer annotation of pages once provided by the now-defunct Third Voice [44] in which users could leave Post-It style notes with comments for other users on a third-party's web page.

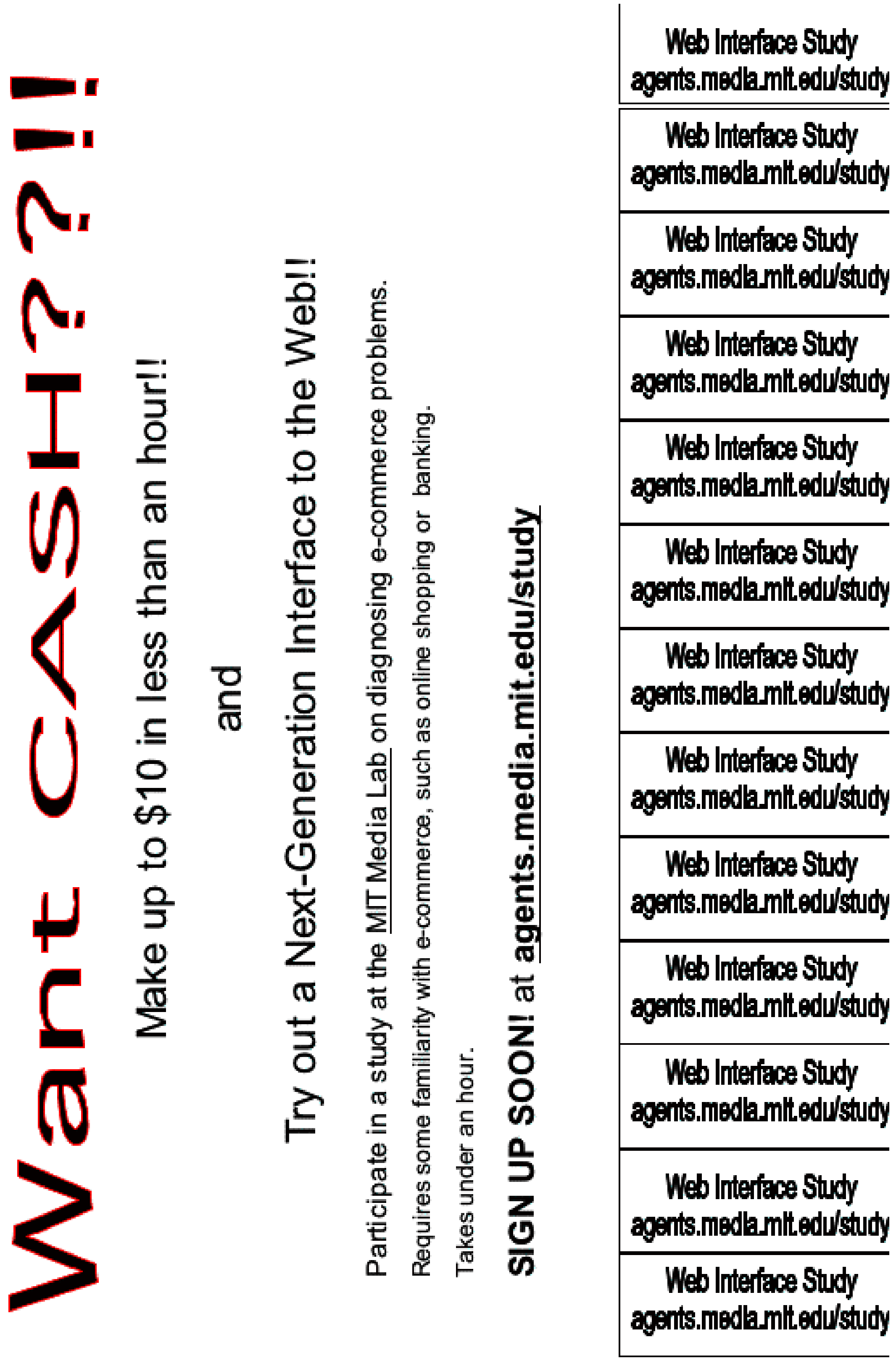
6.12 Visualize Other Systems

Woodstein was designed as a tool for end-users to visualize their web actions. It is general enough to visualize other systems, however. It can be used to inspect the history and status of systems that generate information formattable as web pages, and whose behavior can be described using its simple models. For instance, a system's various logs might be viewed as pages and could be traversed by looking at the semantic relationships among data in different logs, allowing the user to see how a change in one subsystem affected others. With appropriate process descriptions, Woodstein could explain how the different processes that generate the information in the logs occurred.

Appendix A

User Study Recruitment Poster

We recruited participants by posting the flyer in Figure A-1 around the MIT campus.



Want CASH??!!

Make up to $10 in less than an hour!!

and

Try out a Next-Generation Interface to the Web!!

Participate in a study at the MIT Media Lab on diagnosing e-commerce problems.

Requires some familiarity with e-commerce, such as online shopping or banking.

Takes under an hour.

SIGN UP SOON! at **agents.media.mit.edu/study**

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Web Interface Study
agents.media.mit.edu/study

Figure A-1: Poster used to recruit participants.

Appendix B

User Study Introduction and Consent Form

The front of the introduction and consent form signed by participants follows. The back is different for the control and experimental group participants and appears afterwards.

B.1 Introduction and Consent Form Front

Woodstein User Study - Introduction and Consent Form

Woodstein User Study

Introduction

You may be familiar with Clippy, the Microsoft Paperclip. It is an example of a software agent that works in the user interface, though often, it isn't very helpful. We're focused on improving software agents to make them more pro-active in helping people in using the computer, without being so distracting. In particular, we are looking at how an agent working with the web browser can help people better understand what happens with their online actions.

The agent that we will be testing is called Woodstein. It monitors the pages a user visits in order to provide more information about the actions those pages are a part of, particularly when something goes wrong. We will be comparing the effectiveness of the support provided by the agent with the effectiveness of support provided by email-based customer support, an increasingly common way to get help for web transactions.

Consent Form

Participation in this activity is voluntary and completely anonymous. If you are uncomfortable with any question, you may decline to answer it. You may choose to withdraw your consent, and discontinue participation in this activity at any time without prejudice.

We want to find out whether using the software seems natural to users, and we'd like to find out what needs improvment. To determine these things, the experimenter will ask you to describe what you're thinking as you work on the scenario, asking what you expect to see, and what you expect the software to show. We will record what's happening on the screen for these interactions, as well as the audio of the conversation. You may listen to the recording or edit it if you choose. These recordings will only be heard by the head experimenter, Earl Wagner, his assistant, Hyunsuk Kim, and his advisor, Henry Lieberman. They will be deleted when the research is complete or within 6 months.

B.2 Introduction and Consent Form Back for Control Group

The study will consist of the following parts:

1. Introduction, Consent and Pre-Test Questions (5 minutes)
2. Explanation: Walkthrough in Diagnosing Purchase Scenario (5 minutes)
3. Test: Diagnosing Graduation Scenario (20 minutes)
4. Post-Test Questions (10 minutes)

By agreeing to this form you become a consenting participant. You will also have this consent form printed-out. Any inquiries concerning the procedures should be directed to:

Earl Wagner - ewagner@media.mit.edu - 617-253-5334

You may also contact the Chairman of the Committee on the Use of Humans as Experimental Subjects, M.I.T. 253-6787, if you feel you have been treated unfairly as a subject.

I agree to the procedures of this activity

B.3 Introduction and Consent Form Back for Control Group

The study will consist of the following parts:

1. Introduction, Consent and Pre-Test Questions (5 minutes)

2. Explanation: Walkthrough in Tracking and Diagnosing Purchase Scenario Location (20 minutes)
3. Quiz: Diagnosing Purchase Scenario Price (5 minutes)
4. Test: Diagnosing Graduation Scenario (15 minutes)
5. Post-Test Questions (10 minutes)

By agreeing to this form you become a consenting participant. You will also have this consent form printed-out. Any inquiries concerning the procedures should be directed to:

Earl Wagner - ewagner@media.mit.edu - 617-253-5334

You may also contact the Chairman of the Committee on the Use of Humans as Experimental Subjects, M.I.T. 253-6787, if you feel you have been treated unfairly as a subject.

I agree to the procedures of this activity

Appendix C

User Study Explanation Script

C.1 Script for Walk-through

1. The Experimenter asks:

   We're going to see some of how Woodstein works with an example. Woodstein tracks what I'm doing online and matches the steps I take to its models for a processes, like it's model of how a purchase happens. I'm going to go ahead and start making a purchase, to show you how it works. But don't worry, it's only connecting with our simulated web site, Amazin.com

2. The Experimenter loads the order example.

3. The Experimenter says:

   I'm loading the web site for Amazin.com, and the first thing you notice is that Woodstein added a watermark image to the page. That lets me know it's monitoring this page. I can also press down on it to bring up a menu. I want to find out what it knows about what I'm doing right now, so I'll ask it. I'll ask it "what's happenening?".

4. The Experimenter selects "what's happening?" (Figure 2-11).

5. The Experimenter says:

   The agent creates a pop-up window that tells me what it knows about what I'm doing (Figure 2-12). Up here in the grey, I can see a simple description - I saw the Amazin.com front page. It also shows what I'm doing at more and more specific detail. It shows the overall process that I'm triggering, a purchase. It also shows that I'm doing the first part of that - I'm placing an order. The first step of placing an order is browsing. And finally, the first step of that is me loading the Amazin.com front page.

I'm going to go ahead and check out this recommendation it has for me. So I click on it.

6. The Experimenter clicks "The Very Hungry Caterpillar" (Figure 2-13)

7. The Experimenter says:

   You see that the "what's happening" window updates (Figure 2-14). It added some more steps: that I clicked "The Very Hungry Caterpillar", and that I saw the page for "The Very Hungry Caterpillar" as part of finding an item. So already, we can see how it's using it's models of what happens in a purchase process to help organize my action of purchasing something online. This is compared to what you'd get from the browser, a linear history which just shows the list of pages you visited that you can't even go back to to see saved versions of.

   One thing that's helpful that Woodstein does is it saves every page I interact with. You can imagine that's helpful for accessing an old page like an order confirmation. All of these steps are buttons that I can inspect. Just as an example, I can click on the step "You clicks on 'The Very Hungry Caterpillar"', and see the page that Woodstein saved for it. I can even see the exact thing I interacted with, the link that I clicked on. So what we're seeing here is that even while I'm in the middle of an action, in this case making a purchase, I can jump back to previous points and see the pages I interacted with. If you've ever tried clicking on "back" while browsing a site and doing an action like a purchase, you know that sometimes web sites get really confused. This avoids that because rather than clicking on back, I'm just taking a look at a previous page that Woodstein saved.

8. The Experimenter asks if there are any questions and answers them.

9. The Experimenter says:

   OK. Let's take a look at this structured history again. You can see that all the nodes that are deepest in are steps that I took - You saw page, You clicked, You saw page. In addition to saving copies of the pages I interacted with, it also found relevant pages. It found pages related to these abstract processes that it inferred are happening.

I can even click on one of these processes to see the relevant page describing it. Suppose this were the first time I was placing an order and I wanted to find out more about how it works. I could click on the button "You are placing order" to see the page Woodstein retrieved that describes it.

10. The Experimenter clicks "You are placing order at Amazin.com" (Figure 2-15).

11. The Experimenter says:

    OK, great. Here I see that it retrieved a help page that explains how to place an order. Notice that the process I'm looking at right now looks pressed in. That's because that's the button I've selected. It's the one I'm inspecting right now. Also notice that when we have the "saved page" view and the "what's happening" views side-by-side, and I move the mouse over the different steps described in the "saved page" view, the corresponding steps are highlighted in the what's happening view. So I start with "Placing An Order", then I move to "Browsing", then "Finding Items". Down here, when I mouse over "Adding Items to Shopping Cart", I can see on the status bar for the "saved page" view that that's something Woodstein's expecting, but that I haven't done yet.

    We can also click on any of the buttons in this view, so, for instance, if I wanted to inspect the "Browsing" step, I could click that. That loads Woodstein's saved page view for the "Browsing" process, and also highlights the "Browsing" process in the "what's happening" view.

    Let's continue with placing the order. I want to buy this item, so I click "Add to Shopping Cart" in the browsing page. This page from the Amazin.com web server updates to say that the item is in my shopping cart. I can also see what Woodstein found in the "what's happening" view. The first thing we see is that it has added more steps. I can see that it saw me click on the button, and also that it realized that Amazin.com did some things too. It set the shopping cart item, and the shopping cart subtotal. These are steps that Amazin.com took that Woodstein is also tracking. It was able to recognize that I was adding an item to my shopping cart, and go and retrieve the shopping cart page to get the updated list of items and subtotal. We can see that now when I click on "Amazin.com set shopping cart subtotal". Woodstein shows me the page it retrieved and saved with the shopping cart information.

I can also see that there's more information about this order in the "what's happening" view. Over here on the right, underneath "Your Data", I can see that Woodstein is starting to keep track of my data related to this purchase. So far, it knows about the the shopping cart item and subtotal. You can see that for each step by Amazin.com, the result is some data that Woodstein is tracking. Also, the current status of a process is the last data created for it. So in this case, the result of "You added item to shopping cart" is the subtotal. That trickles all the way up to being the status for the purchase.

12. The Experimenter asks if there are any questions and answers them.

13. The Experimenter answers any questions.

    We're going to jump ahead and examine this purchase from the perspective of after it's happened.

14. The Experimenter loads the purchase scenario (Figure 2-1).

15. The Experimenter says:

    This is where we're going to be doing things related to what you'll do for the quiz and test, so I'll turn the mouse over to you now. Imagine we placed the order with Woodstein a while ago and now we're looking at this charge on the credit card and we want to find out more about this purchase. Let's say we want to find out what the item is. Before we saw how Woodstein is able to show what it's tracking. Now we'll see how it lets you inspect data in pages that you want to find out more about. Go ahead and click on the watermark image.

16. The Experimenter says:

    Before we get farther into this example, I'd like to go over a few points for you. When you go on the web, you see data, like prices, quantities and addresses. You also do actions, like clicking on a button to add an item to your shopping cart. The web site also does an action, it goes ahead and adds the item to your shopping cart. You can think of the entire process of making a purchase online as consisting of actions like those, involving data like the items you order, the shipping address, and so on.

Woodstein watches for all of these things, both data and actions, and the pages that they're associated with. When you click on a button to add an item to your shopping cart, it saves the page and keeps track of what you clicked. When the web site updates your shopping cart, Woodstein goes and retrieves a copy of your shopping cart page and keeps track of the website's action. A process takes data items as inputs and creates a data item as a result.

Woodstein has an inspector mode that you turn on by clicking the watermark. It converts all data and names for processes in pages to buttons. Buttons for data items are rectangular, and the buttons for processes are rounded.

Notice how when you move the mouse over the buttons, the status bar at the bottom of the browser window updates.

17. Experimenter points to status bar

    It tells you the name of the data item or process you're looking at. In this page we can see that there are data items like the amount and transaction number. Above them, the label is also a button. Move the mouse over the number for the amount, then the word "amount".

18. Subject moves mouse over price amount and price label.

    As we can see, the number is the data item, and the label stands for the process that sets it. Notice that the process that sets some data has the data name in quotes. A data item is the result of a process.

    The buttons let you inspect the data and processes directly. You let the agent know what you are interested in finding out by interacting with a button. Everything that looks like a button can be inspected. There are two ways to inspect a button. Clicking on it selects it as the button you're inspecting. You can also press down on the mouse button, and a menu with options comes up. Go over to the actual number for the transaction. Press the left mouse button down until a menu comes up, and don't let go yet.

19. Subject presses down on Order # button (Figure 2-2).

Its menu has a few options. You can ask "how it was set" and "why it was set". Each question creates a separate pop-up window, either the "how" view or the "why" view. Let go of the mouse button while over the "why" question.

20. Subject selects "why" question (Figure 2-3).

    The why view shows why something happened by showing where it fits in the overall history of the action. It gives a short explanation at the top grey area. In the display area below, it has the processes on the left and my data on the right.

    On the left are just the process names. On the right, notice how the data items have the name of the data on the left, then a colon, then the value of the data. Here we see that the value for the shipment location is Philadelphia. On the right is my information that's either a result of a process, or the last data item set within it. So, in this case, since the last data item created for my purchase was the shipment location, that's the current status for the whole process.

    Processes involve steps, and, for instance, the first step of making a purchase is placing an order. When the steps of the process are hidden, the box is a plus, like usual on Windows. When they're shown, the box is a minus. Go ahead and click them open and closed. No box means that there isn't any more detail. For right now, click these boxes open.

21. Subject clicks processes opened.

22. The Experimenter says:

    Let's go back to the original order action, what we started to create in the last example. We can access Woodstein's record of the order by inspecting the order action. Let's just inspect the order confirmation, which will take us back to the page where the order confirmation was created. You can inspect another button by clicking on it, so click on the order confirmation button.

23. Subject clicks order confirmation button.

24. The Experimenter says:

    A new view opens up. When ever a process happens, or a data item is set, Woodstein records the page that shows it. Woodstein shows us the saved pages in its saved page view. Here we can see the original order confirmation that resulted just after placing the order, and we can see the item that was ordered.

25. The Experimenter says:

    Great! Let's review. Woodstein keeps track of data items and processes, and the pages in which they appear. We can ask "why" something happened to get its "why" view which shows where the thing fits into the overall history of the action.

Appendix D

User Study Quiz and Test Scripts

D.1 Script for Price Quiz

Now we want you to have a chance to practice using the system. In the quiz scenario, we go back to the credit card charge. Imagine you're looking at this charge and it looks like the price is too high. You thought the book cost less than it says here. Now you want to go back and see how much it was that the book was discounted. To do that, go back to the page for the book before you added it to the shopping cart.

Great! Let's try something new. Let's say we found that the price was wrong. Then we could inspect it and complain. Try selecting "Looks Wrong". That automatically generates an email that we can send. That just goes to our test server, so let's go ahead and send that.

D.2 Script for Graduation Test

In the test scenario, imagine you're a graduate student, here at MIT, and you're getting your master's of science degree. You're getting ready to graduate and you know you've satisfied all the requirements. You've taken all your classes and you took care of your thesis. But you know that sometimes MIT's computers get messed up, or information isn't entered correctly. Now you're looking at your degree audit page and it looks like you're not eligible to graduate. You want to find out why.

When you're dealing with a large organization, you can set up an appointment to talk to somebody, or wait in line. We want to avoid that as much as possible. We'd like to try to narrow down the exact requirement that they messed up on, so we can complain to exactly the right person. Once you've narrowed down the incorrect requirement as much as possible, we'll complain. We don't have to worry about how they'll fix it, we just want to find out the most specific process or data that went wrong. Assume that the same person is responsible for the process of entering the data for the requirement, and the data item itself.

D.3 Acceptable Guidance for Graduation Test

- "Remember we want to narrow down the requirement as much as possible."
- "Assume that the same person is responsible for the process of entering the data for the requirement, and the data item itself."

Appendix E

Frequently Asked Questions about Woodstein

1. Q. Why would anyone ever want to "debug" e-commerce processes?

   A. Have you ever looked at an unfamiliar charge on your credit card and wanted to know what item it paid for, or where the item is now? Either of these can be found within two clicks after turning on Woodstein's inspector - see section 2.2 on page 25.

2. Q. But it doesn't sound like Woodstein helps with debugging, it just sounds like it helps with problem diagnosis.

   A. Indeed, the focus of Woodstein so far has been on problem diagnosis; see section 3.6 on page 74. Although we've implemented good support for diagnosing problems, the support for resolving problems is less robust and is an area for future work.

3. Q. Yeah, but is Woodstein really something that people would want to use?

   A. A majority of the people we asked who used Woodstein agreed that they'd like to have it to help with their own credit card transactions and problems that arise. In fact, half of the people we asked *strongly agreed* that they'd like to use it. See section 5.3.3 on page 112.

4. Q. What about security? Won't it be a bad idea for an agent to be saving my passwords and account info?

   A. Woodstein is implemented as a web proxy that runs on your computer; see section 4.3.2 on page 80. Saving information with it is no worse than saving user-names and passwords for your accounts with Internet Explorer or Netscape.

5. Q. What does the agent do exactly? Will it buy anything or do anything else I don't want?

   A. The agent only gathers information by logging into accounts, checking pages, and so on. It just retrieves pages in creating a detailed record for your action, and provide an integrated view of your data even when they span multiple sites. It only retrieves data when it wants to check to make sure that the web site is reacting to your actions appropriately (see section 4.6 on page 94). For example, when you add an item to your shopping cart, it makes sure the web site updates the shopping cart correctly.

6. Q. Where are the descriptions of the processes located?

   A. They are located on the your computer, as is the agent. If this were a product, the agent could receive updated models for processes and pages either as you browse, or at periodic intervals.

7. Q. Does it take up a lot of space to store all those web pages?

   A. Not really, especially if they're just saved as text. A larger page, such as an item description page on the actual Amazon.com site might be around 50 KB. A megabyte of disk can hold 20 pages like those, and a gigabyte can hold 20,000. If it saved just the pages the user interacts with securely (via secure http, https), it would take a long time to fill a gigabyte.

Bibliography

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1988.

[2] Thomas Ball and Stephen G. Eick. Visualizing program slices. In *IEEE/CS Symposium on Visual Languages*, pages 288–295, 1994.

[3] Rob Barrett and Paul P. Maglio. Intermediaries: new places for producing and manipulating web content. In *Seventh International World Wide Web Conference*, 1998.

[4] Rob Barrett, Paul P. Maglio, and Daniel C. Kellem. WBI: A confederation of agents that personalize the web. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 496–499, New York, 1997. ACM Press.

[5] Carl Bernstein and Bob Woodward. *All the President's Men*. Simon and Schuster, 1974.

[6] Peter Brusilovsky. Program visualization as a debugging tool for novices. In *INTERACT '93 and CHI '93 conference companion on Human factors in computing systems*, pages 29–30. ACM Press, 1993.

[7] J. Budzik and K.J. Hammond. User interactions with everyday applications as context for just-in-time information access. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, New Orleans, Louisiana, 2000. ACM Press.

[8] Sandra Carberry. Plan recognition: Achievements, problems, and prospects.

[9] Philip R. Cohen, C. Raymond Perrault, and James F. Allen. Beyond question answering. In Wendy G. Lehnert and Martin H. Ringle, editors, *Strategies for Natural Language Processing*, pages 245–274. Lawrence Erlbaum Associates, 1982.

[10] James S. Collofello and S. N. Woodfield. Evaluating the effectiveness of reliability-assurance techniques. *Journal of Systems and Software*, 9(3):191–195, 1989.

[11] Microsoft Corp. Internet Explorer.

[12] Microsoft Corp. .NET Passport.

[13] Netscape Communications Corp. Navigator.

[14] David Daniels and David Schatsky. Quantifying the cost of poor service: Investing in customer service to defend revenues, April 2002.

[15] Randall Davis. Teiresias: Applications of meta-level knowledge. In Randall Davis and Douglas B. Lenat, editors, *Knowledge-Based Systems in Artificial Intelligence*, pages 225–484. McGraw-Hill, New York, 1982.

[16] John Domingue, Martin Dzbor, and Enrico Motti. Semantic layering with magpie. In *Twelfth International World Wide Web Conference*, 2003.

[17] Paul Dourish, Annette Adler, and Brian Cantwell Smith. Organising user interfaces around reflective accounts. In Gregor Kiczales, editor, *Proceedings of Reflection 96*, pages 235–244, April 1996.

[18] Adele Goldberg and David Robson. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley, 1983.

[19] Barbara Grosz and Candice Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12, 1986.

[20] Stephen Hawking. *A Brief History of Time - from the Big Bang to Black Holes.* Gradiva, 1988.

[21] Tommy Hoffner, Mariam Kamkar, and Peter Fritzson. Evaluation of program slicing tools. In *Automated and Algorithmic Debugging*, pages 51–69, 1995.

[22] TOWER Technology Inc. Webcapture.

[23] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: The story of squeak, a practical smalltalk written in itself. In *Proceedings OOPSLA'97*, pages 318–326. ACM Press, November 1997.

[24] Bogden Korel and Janusz W. Laski. Dynamic program slicing. *Information Processing Letters*, 29(10):155–163, 1988.

[25] Bruce Krulwich. Automating the internet: agents as user surrogates. *IEEE Internet Computing*, 1(4):34–38, 1997.

[26] Nicholas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–735, 1997.

[27] Henry Lieberman. Letizia: An agent that assists web browsing. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[28] Henry Lieberman. Autonomous Interface Agents. In *Proceedings of the ACM Conference on Computers and Human Interface, CHI-97*, Atlanta, Georgia, 1997.

[29] Henry Lieberman. Introduction and guest editor, special issue on the debugging scandal. *Communications of the ACM*, 40(4):26–29, 1997.

[30] Henry Lieberman, editor. *Your Wish is My Command: Giving Users the Power to Instruct their Software.* Morgan Kaufmann, 2001.

[31] Henry Lieberman and Christopher Fry. Zstep 95: A reversible, animated source code stepper. In John Stasko, John Domingue, Mark Brown, and Blaine Price, editors, *Software Visualization: Programming as a Multimedia Experience.* MIT Press, Cambridge, MA, 1997.

[32] Henry Lieberman and Ted Selker. Out of context: Computer systems that learn about, and adapt to, context. *IBM Systems Journal*, 39(3–4):617–634, 2000.

[33] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[34] Ted H. Nelson. What's on my mind: An invited talk at the first wearable computer conference. 'what note?', May 1998.

[35] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, 1980.

[36] Liberty Alliance Project. www.projectliberty.org.

[37] Ramana Rao. Implementation reflection in silica. In *European Conference on Object-Oriented Programming*, pages 251–267. Springer Verlag, 1991.

[38] Bradley J. Rhodes. Margin notes: building a contextually aware associative memory. In *Proceedings of the 5th international conference on Intelligent user interfaces*, pages 219–224. ACM Press, 2000.

[39] Charles Rich, Candice Sidner, , and Neal Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15–25, 2001.

[40] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main. End-user software visualizations for fault localization. In *ACM Symposium on Software Visualization*, 2003.

[41] Brian C. Smith. Reflection and semantics in lisp. In *11th ACM Symposium on Principles of Programming Languages*, pages 23–35, 1984.

[42] Richard Stallman. *GNU Emacs Manual*. Free Software Foundation Inc., Cambridge, MA, 1986.

[43] David Ungar, Henry Lieberman, and Christopher Fry. Debugging and the experience of immediacy. *Communications of the ACM*, 40(4):38–43, 1997.

[44] Third Voice.

[45] Daniel Weinreb and David Moon. *The Lisp Machine Manual*. Symbolics Inc., 1981.

[46] Mark Weiser. Program slicing. In *Proceedings of the Fifth International Conference on Software Engineering*, pages 439–449, New York, 1981. IEEE.

[47] Aaron Wilson, Margaret Burnett, Laura Beckwith, Orion Granatir, Ledah Casburn, Curtis Cook, Mike Durham, and Gregg Rothermel. Harnessing curiosity to increase

correctness in end-user programming. In *ACM Conference on Human Factors in Computing Systems*, 2003.